Synapse Documentation

Release 2.167.0

The Vertex Project

Apr 19, 2024

CONTENTS:

1 Introduction **3**
- 1.1 Key Features . . . 3
- 1.2 What's Next? . . . 5

2 Getting Started **7**
- 2.1 Synapse Quickstart . . . 7
- 2.2 Open-Source Synapse . . . 8
- 2.3 Synapse Demo Instance . . . 8

3 Synapse User Guide **11**
- 3.1 Background . . . 11
- 3.2 Data Model . . . 15
- 3.3 Analytical Model . . . 26
- 3.4 Views and Layers . . . 33
- 3.5 Tools . . . 37
- 3.6 Storm Reference . . . 60
- 3.7 Storm Advanced . . . 319

4 Synapse Admin Guide **361**
- 4.1 Enable Synapse Power-Ups . . . 361
- 4.2 Create and Manage Users and Roles . . . 362
- 4.3 Assign and Manage Permissions . . . 366
- 4.4 Add Extended Model Elements . . . 394
- 4.5 Manage Model Deprecations . . . 396
- 4.6 Configure a Mirrored Layer . . . 397

5 Synapse Deployment Guide **399**
- 5.1 Introduction . . . 399
- 5.2 Prepare your Hosts . . . 399
- 5.3 Decide on a Name . . . 400
- 5.4 Deploy AHA Service . . . 400
- 5.5 Deploy Axon Service . . . 401
- 5.6 Deploy JSONStor Service . . . 403
- 5.7 Deploy Cortex Service . . . 403
- 5.8 Deploy Cortex Mirror (optional) . . . 404
- 5.9 Enroll CLI Users . . . 405
- 5.10 Configure a Storm Query Pool (optional) . . . 406
- 5.11 What's next? . . . 407

6 Synapse Devops Guide **409**
- 6.1 Overview . . . 409

6.2 Common Devops Tasks . . . 410
6.3 Synapse Services . . . 425
6.4 Devops Details . . . 440

7 Synapse Developer Guide **483**
7.1 Rapid Power-Up Development . . . 483
7.2 Synapse Architecture . . . 492
7.3 Cortex Development Quickstart . . . 494
7.4 Synapse Docker Builds . . . 497
7.5 Storm Service Development . . . 499
7.6 Storm API Guide . . . 506

8 Synapse Glossary **521**
8.1 A . . . 521
8.2 B . . . 522
8.3 C . . . 522
8.4 D . . . 525
8.5 E . . . 526
8.6 F . . . 528
8.7 G . . . 530
8.8 H . . . 531
8.9 I . . . 531
8.10 K . . . 532
8.11 L . . . 532
8.12 M . . . 533
8.13 N . . . 534
8.14 O . . . 535
8.15 P . . . 535
8.16 Q . . . 538
8.17 R . . . 538
8.18 S . . . 540
8.19 T . . . 542
8.20 U . . . 546
8.21 V . . . 546
8.22 W . . . 547

9 Synapse Contributors Guide **549**
9.1 Contributing to Synapse . . . 549
9.2 Synapse Doc Mastering . . . 557
9.3 Synapse Release Process . . . 565

10 Synapse Python API **569**
10.1 synapse package . . . 569

11 Synapse HTTP/REST API **945**
11.1 HTTP/REST API Conventions . . . 945
11.2 Authentication . . . 945
11.3 Cortex . . . 951
11.4 Aha . . . 958
11.5 Axon . . . 960

12 Synapse Data Model **963**
12.1 Synapse Data Model - Types . . . 963
12.2 Synapse Data Model - Forms . . . 1083
12.3 Datamodel Deprecation Policy . . . 1804

13 Storm Library Documentation **1807**
13.1 Storm Libraries 1807
13.2 Storm Types 1908

14 Synapse Power-Ups **1993**
14.1 Rapid Power-Ups 1993
14.2 Advanced Power-Ups 1994

15 Synapse User Interface **1995**

16 Synapse Support **1997**
16.1 Slack 1997
16.2 Service Desk 1997

17 Synapse Changelog **1999**
17.1 v2.167.0 - 2024-04-19 1999
17.2 v2.166.0 - 2024-04-05 2001
17.3 v2.165.0 - 2024-03-25 2003
17.4 v2.164.0 - 2024-03-01 2007
17.5 v2.163.0 - 2024-02-21 2008
17.6 v2.162.0 - 2024-02-15 2008
17.7 v2.161.0 - 2024-02-06 2011
17.8 v2.160.0 - 2024-01-24 2012
17.9 v2.159.0 - 2024-01-16 2014
17.10 v2.158.0 - 2024-01-03 2015
17.11 v2.157.0 - 2023-12-21 2015
17.12 v2.156.0 - 2023-12-08 2016
17.13 v2.155.0 - 2023-11-17 2018
17.14 v2.154.1 - 2023-11-15 2019
17.15 v2.154.0 - 2023-11-15 2019
17.16 v2.153.0 - 2023-10-27 2021
17.17 v2.152.0 - 2023-10-17 2022
17.18 v2.151.0 - 2023-10-06 2024
17.19 v2.150.0 - 2023-09-22 2025
17.20 v2.149.0 - 2023-09-14 2026
17.21 v2.148.0 - 2023-09-05 2027
17.22 v2.147.0 - 2023-08-31 2028
17.23 v2.146.0 - 2023-08-29 2028
17.24 v2.145.0 - 2023-08-25 2028
17.25 v2.144.0 - 2023-08-09 2031
17.26 v2.143.0 - 2023-07-28 2032
17.27 v2.142.2 - 2023-07-19 2033
17.28 v2.142.1 - 2023-07-19 2033
17.29 v2.142.0 - 2023-07-19 2034
17.30 v2.141.0 - 2023-07-07 2036
17.31 v2.140.1 - 2023-06-30 2037
17.32 v2.140.0 - 2023-06-30 2037
17.33 v2.139.0 - 2023-06-16 2038
17.34 v2.138.0 - 2023-06-13 2039
17.35 v2.137.0 - 2023-06-09 2039
17.36 v2.136.0 - 2023-06-02 2042
17.37 v2.135.0 - 2023-05-24 2043
17.38 v2.134.0 - 2023-05-17 2043
17.39 v2.133.1 - 2023-05-09 2044
17.40 v2.133.0 - 2023-05-08 2044

17.41 v2.132.0 - 2023-05-02 . . . 2047
17.42 v2.131.0 - 2023-05-02 . . . 2047
17.43 v2.130.2 - 2023-04-26 . . . 2049
17.44 v2.130.1 - 2023-04-25 . . . 2049
17.45 v2.130.0 - 2023-04-25 . . . 2049
17.46 v2.129.0 - 2023-04-17 . . . 2050
17.47 v2.128.0 - 2023-04-11 . . . 2051
17.48 v2.127.0 - 2023-04-05 . . . 2052
17.49 v2.126.0 - 2023-03-30 . . . 2053
17.50 v2.125.0 - 2023-03-14 . . . 2054
17.51 v2.124.0 - 2023-03-09 . . . 2055
17.52 v2.123.0 - 2023-02-22 . . . 2056
17.53 v2.122.0 - 2023-01-27 . . . 2059
17.54 v2.121.1 - 2022-01-23 . . . 2060
17.55 v2.121.0 - 2022-01-20 . . . 2061
17.56 v2.120.0 - 2023-01-11 . . . 2062
17.57 v2.119.0 - 2023-01-09 . . . 2062
17.58 v2.118.0 - 2023-01-06 . . . 2063
17.59 v2.117.0 - 2023-01-04 . . . 2064
17.60 v2.116.0 - 2022-12-14 . . . 2066
17.61 v2.115.1 - 2022-12-02 . . . 2067
17.62 v2.115.0 - 2022-12-01 . . . 2067
17.63 v2.114.0 - 2022-11-15 . . . 2068
17.64 v2.113.0 - 2022-11-04 . . . 2068
17.65 v2.112.0 - 2022-10-18 . . . 2072
17.66 v2.111.0 - 2022-10-12 . . . 2072
17.67 v2.110.0 - 2022-10-07 . . . 2073
17.68 v2.109.0 - 2022-09-27 . . . 2074
17.69 v2.108.0 - 2022-09-12 . . . 2074
17.70 v2.107.0 - 2022-09-01 . . . 2075
17.71 v2.106.0 - 2022-08-23 . . . 2076
17.72 v2.105.0 - 2022-08-19 . . . 2077
17.73 v2.104.0 - 2022-08-09 . . . 2078
17.74 v2.103.0 - 2022-08-05 . . . 2079
17.75 v2.102.0 - 2022-07-25 . . . 2081
17.76 v2.101.1 - 2022-07-14 . . . 2083
17.77 v2.101.0 - 2022-07-12 . . . 2083
17.78 v2.100.0 - 2022-06-30 . . . 2084
17.79 v2.99.0 - 2022-06-23 . . . 2085
17.80 v2.98.0 - 2022-06-17 . . . 2085
17.81 v2.97.0 - 2022-06-06 . . . 2086
17.82 v2.96.0 - 2022-05-31 . . . 2086
17.83 v2.95.1 - 2022-05-24 . . . 2087
17.84 v2.95.0 - 2022-05-24 . . . 2087
17.85 v2.94.0 - 2022-05-18 . . . 2088
17.86 v2.93.0 - 2022-05-04 . . . 2089
17.87 v2.92.0 - 2022-04-28 . . . 2091
17.88 v2.91.1 - 2022-04-24 . . . 2091
17.89 v2.91.0 - 2022-04-21 . . . 2091
17.90 v2.90.0 - 2022-04-04 . . . 2092
17.91 v2.89.0 - 2022-03-31 . . . 2093
17.92 v2.88.0 - 2022-03-23 . . . 2093
17.93 v2.87.0 - 2022-03-18 . . . 2094
17.94 v2.86.0 - 2022-03-09 . . . 2095

17.95 v2.85.1 - 2022-03-03 2096
17.96 v2.85.0 - 2022-03-03 2097
17.97 v2.84.0 - 2022-02-22 2098
17.98 v2.83.0 - 2022-02-17 2098
17.99 v2.82.1 - 2022-02-11 2099
17.100 v2.82.0 - 2022-02-10 2099
17.101 v2.81.0 - 2022-01-31 2100
17.102 v2.80.1 - 2022-01-26 2100
17.103 v2.80.0 - 2022-01-25 2100
17.104 v2.79.0 - 2022-01-18 2101
17.105 v2.78.0 - 2022-01-14 2101
17.106 v2.77.0 - 2022-01-07 2102
17.107 v2.76.0 - 2022-01-04 2102
17.108 v2.75.0 - 2021-12-16 2103
17.109 v2.74.0 - 2021-12-08 2104
17.110 v2.73.0 - 2021-12-02 2105
17.111 v2.72.0 - 2021-11-23 2105
17.112 v2.71.1 - 2021-11-22 2106
17.113 v2.71.0 - 2021-11-19 2106
17.114 v2.70.1 - 2021-11-08 2107
17.115 v2.70.0 - 2021-11-03 2107
17.116 v2.69.0 - 2021-11-02 2108
17.117 v2.68.0 - 2021-10-29 2108
17.118 v2.67.0 - 2021-10-27 2108
17.119 v2.66.0 - 2021-10-26 2109
17.120 v2.65.0 - 2021-10-16 2109
17.121 v2.64.1 - 2021-10-08 2110
17.122 v2.64.0 - 2021-10-06 2110
17.123 v2.63.0 - 2021-09-29 2111
17.124 v2.62.1 - 2021-09-22 2111
17.125 v2.62.0 - 2021-09-21 2112
17.126 v2.61.0 - 2021-09-17 2112
17.127 v2.60.0 - 2021-09-07 2113
17.128 v2.59.0 - 2021-09-02 2113
17.129 v2.58.0 - 2021-08-26 2114
17.130 v2.57.0 - 2021-08-24 2114
17.131 v2.56.0 - 2021-08-19 2115
17.132 v2.55.0 - 2021-08-18 2115
17.133 v2.54.0 - 2021-08-05 2115
17.134 v2.53.0 - 2021-08-05 2116
17.135 v2.52.1 - 2021-07-30 2118
17.136 v2.52.0 - 2021-07-29 2118
17.137 v2.51.0 - 2021-07-26 2118
17.138 v2.50.0 - 2021-07-22 2119
17.139 v2.49.0 - 2021-07-19 2119
17.140 v2.48.0 - 2021-07-13 2120
17.141 v2.47.0 - 2021-07-07 2120
17.142 v2.46.0 - 2021-07-02 2120
17.143 v2.45.0 - 2021-06-25 2121
17.144 v2.44.0 - 2021-06-23 2121
17.145 v2.43.0 - 2021-06-21 2122
17.146 v2.42.2 - 2021-06-11 2123
17.147 v2.42.1 - 2021-06-09 2123
17.148 v2.42.0 - 2021-06-03 2123

17.149 v2.41.1 - 2021-05-27 . . . 2124
17.150 v2.41.0 - 2021-05-27 . . . 2124
17.151 v2.40.0 - 2021-05-26 . . . 2124
17.152 v2.39.1 - 2021-05-21 . . . 2125
17.153 v2.39.0 - 2021-05-20 . . . 2125
17.154 v2.38.0 - 2021-05-14 . . . 2126
17.155 v2.37.0 - 2021-05-12 . . . 2126
17.156 v2.36.0 - 2021-05-06 . . . 2127
17.157 v2.35.0 - 2021-04-27 . . . 2127
17.158 v2.34.0 - 2021-04-20 . . . 2128
17.159 v2.33.1 - 2021-04-13 . . . 2128
17.160 v2.33.0 - 2021-04-12 . . . 2128
17.161 v2.32.1 - 2021-04-01 . . . 2129
17.162 v2.32.0 - 2021-03-30 . . . 2129
17.163 v2.31.1 - 2021-03-25 . . . 2130
17.164 v2.31.0 - 2021-03-24 . . . 2130
17.165 v2.30.0 - 2021-03-17 . . . 2130
17.166 v2.29.0 - 2021-03-11 . . . 2131
17.167 v2.28.1 - 2021-03-08 . . . 2131
17.168 v2.28.0 - 2021-02-26 . . . 2132
17.169 v2.27.0 - 2021-02-16 . . . 2132
17.170 v2.26.0 - 2021-02-05 . . . 2133
17.171 v2.25.0 - 2021-02-01 . . . 2133
17.172 v2.24.0 - 2021-01-29 . . . 2134
17.173 v2.23.0 - 2021-01-21 . . . 2134
17.174 v2.22.0 - 2021-01-19 . . . 2135
17.175 v2.21.1 - 2021-01-04 . . . 2135
17.176 v2.21.0 - 2020-12-31 . . . 2135
17.177 v2.20.0 - 2020-12-29 . . . 2136
17.178 v2.19.0 - 2020-12-27 . . . 2136
17.179 v2.18.1 - 2020-12-24 . . . 2136
17.180 v2.18.0 - 2020-12-23 . . . 2137
17.181 v2.17.1 - 2020-12-22 . . . 2137
17.182 v2.17.0 - 2020-12-22 . . . 2137
17.183 v2.16.1 - 2020-12-17 . . . 2138
17.184 v2.16.0 - 2020-12-15 . . . 2138
17.185 v2.15.0 - 2020-12-11 . . . 2139
17.186 v2.14.2 - 2020-12-10 . . . 2139
17.187 v2.14.1 - 2020-12-09 . . . 2139
17.188 v2.14.0 - 2020-12-09 . . . 2140
17.189 v2.13.0 - 2020-12-04 . . . 2140
17.190 v2.12.3 - 2020-12-03 . . . 2140
17.191 v2.12.2 - 2020-12-01 . . . 2140
17.192 v2.12.1 - 2020-12-01 . . . 2141
17.193 v2.12.0 - 2020-11-30 . . . 2141
17.194 v2.11.0 - 2020-11-25 . . . 2141
17.195 v2.10.2 - 2020-11-20 . . . 2142
17.196 v2.10.1 - 2020-11-17 . . . 2142
17.197 v2.10.0 - 2020-11-17 . . . 2142
17.198 v2.9.2 - 2020-10-27 . . . 2143
17.199 v2.9.1 - 2020-10-22 . . . 2144
17.200 v2.9.0 - 2020-10-19 . . . 2144
17.201 v2.8.0 - 2020-09-22 . . . 2146
17.202 v2.7.3 - 2020-09-16 . . . 2146

17.203 v2.7.2 - 2020-09-04 2147
17.204 v2.7.1 - 2020-08-26 2147
17.205 v2.7.0 - 2020-08-21 2148
17.206 v2.6.0 - 2020-08-13 2149
17.207 v2.5.1 - 2020-08-05 2149
17.208 v2.5.0 - 2020-07-30 2150
17.209 v2.4.0 - 2020-07-15 2151
17.210 v2.3.1 - 2020-07-13 2151
17.211 v2.3.0 - 2020-07-09 2151
17.212 v2.2.2 - 2020-07-03 2152
17.213 v2.2.1 - 2020-06-30 2152
17.214 v2.2.0 - 2020-06-26 2153
17.215 v2.1.2 - 2020-06-18 2153
17.216 v2.1.1 - 2020-06-16 2153
17.217 v2.1.0 - 2020-06-16 2154
17.218 v2.0.0 - 2020-06-08 2154
17.219 API Deprecation Notice - 2023-10-01 2154

18 Indices and tables **2157**

Python Module Index **2159**

Index **2163**

Star us on GitHub | Watch Synapse 101

CHAPTER

ONE

INTRODUCTION

Synapse is a versatile central intelligence and analysis system created to support analyst teams in every stage of the intelligence life cycle.

The Vertex Project designed and developed Synapse to help analysts and algorithms answer complex questions which require the fusion of large data sets from disparate sources that span multiple disciplines.

Synapse's data store (known as a *Cortex*) is organized as a hypergraph. Combined with its structured and extensible *Data Model* and the powerful and intuitive *Storm* query language, Synapse gives analysts unparalleled power and flexibility to ask and answer any question, even over large and complex data sets.

Note: A *Cortex* may easily grow to billions of nodes, but is not designed to consume and create billions of nodes per day. In other words, Synapse is not meant to replace your big-data/data-lake storage; Synapse is designed to connect to your data sources on demand in order to ingest data relevant for your analysis into the Synapse intelligence platform.

1.1 Key Features

Extensible Data Model

Synapse includes an extensive (and extensible) *Data Model* capable of representing real-world objects, relationships, and events in an intuitive and realistic manner.

Strong Typing

Synapse uses *Type Normalization* and *Type Enforcement* to apply meaningful constraints to data to ensure it is well-formed, preventing "bad data" from cluttering the knowledge store. *Type Awareness* simplifies use of the Storm query language and helps analysts discover novel relationships in the data.

Powerful and Intuitive Query Language

Synapse's *Storm* query language is a powerful, intuitive "data language" used to interact with data in a Synapse Cortex. Storm frees analysts from the limitations of "canned" queries or hard-coded data navigation and allows them to ask - and answer - **any** analytical question.

Unified Analysis Platform

Synapse's unified data store provides analysts with a shared view into the same set of data and analytical annotations, allowing them to better coordinate, collaborate, and peer-review their work.

Designed and Tested in Partnership with Analysts

Synapse is the product of a unique close collaboration between Vertex developers and analysts that leverages innovative software design and engineering to directly support analyst needs and workflows.

Modular Architecture

Synapse is extensible through **Power-Ups** (see *Power-Up*) that add functionality, integrate with third-party data sources, or connect to external databases.

Record Analytical Assessments

Synapse allows analysts to annotate data with assessments and observations through a flexible and extensible set of tags (see *Tag*). By recording assessments **and** data in a structured manner, analysts and algorithms can leverage **both** in their queries and workflows.

"Git for Analysis"

Synapse supports the use of layers (see *Layer*) to comprise a *View* into Synapse's data store. Analysts can create a *Fork* of a given view and use it for testing or research without modifying the underlying production data. Once work in the fork is complete, changes can be merged into the production view or discarded.

Fine-Grained Access Controls

Synapse provides access controls and detailed permissions that can be applied to users or roles. Permissions can be specified broadly or to a level of detail that restricts a user to setting a single property on a single form.

Flexible Automation

Synapse allows you to create custom automation for both analytical and administrative tasks, ensuring consistency and eliminating tedious or time-consuming workflows. Automation (see *Storm Reference - Automation*) is provided using event-based triggers (*Trigger*), scheduled cron jobs, or stored macros.

API Access

Synapse includes multiple well-documented APIs for interacting with the data store and other Synapse components. (See *Synapse HTTP/REST API* and *Synapse Python API*.)

Lightning Fast Performance

Synapse uses LMDB for high-performance key-value indexing and storage, combined with asynchronous, streaming processing. This means queries start returning results as soon as they are available - so your "time to first node" is typically milliseconds, regardless of the size of your result set.

Horizontally and Vertically Scalable

A single Synapse Cortex can easily scale vertically to hold tens of billions of nodes. In addition, Synapse supports high-availability topologies such as mirroring.

1.2 What's Next?

Get Started!	• There are several options for you to deploy and start using Synapse! See our *Getting Started* guide to see which one is right for you. • Watch Synapse 101
Users	• *Synapse User Guide* • *Storm Reference* • Changelog • Ask a question in Slack
DevOps	• *Synapse Devops Guide* • *Synapse Deployment Guide* • Synapse sizing guide
Developers	• *Synapse Developer Guide* • *Synapse HTTP/REST API* • *Synapse Python API* • *Synapse Data Model* • *Storm Library Documentation*
Admins	• *Synapse Admin Guide*
Synapse UI (commercial)	• Synapse UI ("Optic") documentation (includes guides for users, devops, and developers)
Learn More	• Upcoming Webinars • Video Library • Visit The Vertex Project Website
Connect With Us!	• Slack • Twitter • LinkedIn • "Star" us on Github

CHAPTER

TWO

GETTING STARTED

So you've looked over our *Introduction* to Synapse and want to try it out! What do you do next?

Open-source Synapse and demo versions of commercial Synapse (Synapse Enterprise) are both available for you to deploy and test. Both versions include the **same key features,** including Synapse's core architecture and functionality, our extensive data model, and the full capabilities of the Storm query language and libraries.

Open-source versions of Synapse provide a **command-line interface** (the Storm CLI) to interact with Synapse and its data. You can download *Open-Source Synapse* from our Github repository or use *Synapse Quickstart* to easily load a basic instance of Synapse.

Demo instances of Synapse Enterprise include Synapse's **web-based UI**, also known as **Optic.**

- If you want to get started with Synapse as quickly as possible, then a *Synapse Demo Instance* or *Synapse Quickstart* are right for you.
- If you're interested in deploying your own test or production environment, then take a look at *Open-Source Synapse*.

We'll explain each option in more detail below.

2.1 Synapse Quickstart

Synapse Quickstart is a Docker container that includes everything you need to start using Synapse and the Storm CLI right away. Because Synapse Quickstart is self-contained, you can easily install and launch this basic Synapse instance on Linux, Windows, or MacOS.

You can find the instructions to download and install Synapse Quickstart here.

Synapse Quickstart is best for:

- Individual users.
- Users who want to test Synapse without the need for a formal deployment.
- Users who are most interested in learning about Synapse's data and analytical models and the Storm query langauge (vs. deployment or development tasks).
- Users who want to test or use Synapse with proprietary or sensitive data that must be hosted locally.

Synapse Quickstart is **not** pre-loaded with any data.

2.2 Open-Source Synapse

The full open-source version of Synapse is available from our Github repository. Instructions for deploying a test or production environment are available in the *Synapse Deployment Guide*.

Open-source Synapse is best for:

- Users who want to work with or try out a full version of Synapse.
- Supporting multiple users and / or networked users, including the (optional) ability to configure roles and permissions.
- Developers who want to build on or integrate with Synapse.
- Users who want to test or use Synapse with proprietary or sensitive data that must be hosted locally.

Open-source Synapse is **not** pre-loaded with any data. However, some of Synapse's Power-Ups are available as open source and can help you automate adding data to Synapse:

- Synapse-MISP
- Synapse-MITRE-ATTACK
- Synapse-TOR

2.3 Synapse Demo Instance

Commercial Synapse (Synapse Enterprise) and our commercial demo instances include the web-based Synapse UI (Optic). **Demo instances** are **cloud-hosted,** so there is nothing for you configure or deploy to get started - all you need is a web browser (we recommend Chrome).

You can request a demo instance from our web site.

Note: Synapse Enterprise can be deployed either on premises or in the cloud. Only the demo instances are cloud-only.

Demo instances provide access to all of Synapse's Rapid Power-Ups, both open-source and commercial. Any Rapid Power-Up can be installed in your demo instance (although some Power-Ups may reqiure API keys and / or paid subscriptions from the associated third-party).

Demo instances are updated automatically each week with any new releases of Synapse and Optic. New or updated Rapid Power-Ups are available upon release and can be updated manually from the Power-Ups Tool.

In addition, demo instances are **pre-loaded** with sample data and tags (just under 300,000 objects). You can explore the data on your own, or use our APT1 Scavenger Hunt as a guided way to learn about the Synapse UI and Storm query language.

A **demo instance** is best for:

- Users who want to test all of Synapse's features and capabilities, including those only available with Synapse Enterprise.
- Supporting multiple users and / or networked users, including the (optional) ability to configure roles and permissions.
- Simple deployment - no hardware/software needed (other than a web browser).
- Developers who want insight into developing Power-Ups or Workflows.
- Users and developers who want access to the "latest and greatest" releases and features during testing.

- Users who want to take advantage of all of Synapse's features (including built-in Help for Synapse's data model, Storm auto-complete, etc.) while learning - even if you ultimately deploy an open-source version.

Note: Because demo instances are cloud-based, they are **not suitable** for hosting any sensitive or proprietary data.

CHAPTER

THREE

SYNAPSE USER GUIDE

This User Guide is written by and for Synapse users and is intended to provide a general overview of Synapse concepts and operations. Technical documentation appropriate for Synapse deployment and development can be found elsewhere in the Document Index.

The User Guide is a living document and will continue to be updated and expanded as appropriate. The current sections are:

3.1 Background

3.1.1 Why Synapse?

Synapse is a versatile central intelligence and analysis system created to support analyst teams in every stage of the intelligence life cycle. We designed Synapse to answer complex questions which require the fusion of large data sets from a broad range of sources that span multiple disciplines. Analysis is based on representing all data in a structured model that allows analysts or algorithms to query, annotate, navigate, and reason over the collected data.

Tip: See Synapse's *Key Features* for an overview of Synapse's advantages!

Synapse is based on a proven methodology informed by real-world experience. Synapse grew out of the need to track a complex, diverse, and very large data set: namely, cyber threat data. Synapse is the successor to the proprietary, directed graph-based analysis platform (Nucleus) used within Mandiant to produce the APT1 Report.

Synapse and its predecessors were designed from the beginning to support the following critical elements:

- The use of a **shared analytical workspace** to give analysts access to the same data and assessments in real time.
- The principle that **relationships among and conclusions about data should be self-evident.** That is, to the extent possible, data and analytical findings must be represented so that analysis captured within the system should "speak for itself".

These features give Synapse the following advantages:

- Synapse allows (and requires) analysts to "show their work" in a reasonably concise manner. Analysts should not have to refer to long-form reporting (or rely on the unquestioned word of a subject matter expert) to trace a line of analytical reasoning.
- Synapse allows analysts to better review and validate their findings. Conflicting analysis is highlighted through the structure of the data itself. Analysis can readily be questioned, reviewed, deconflicted, and ultimately improved.

- Because Synapse's knowledge store is continually expanded, updated, and revised, it always represents the current, combined understanding of its data and analysis. Unlike prose reports or tickets, Synapse is never stale or outdated.

Synapse's hypergraph design addresses many of the shortcomings we identified with earlier directed graph and prototype hypergraph systems. In addition, because our experience taught us the power of a flexible analysis platform over any large and disparate data set, Synapse has been designed to be flexible, modular, and adaptable to **any** knowledge domain - not just cyber threat data.

Many of the real-world examples in this User Guide reference data from the fields of information technology or cyber threat intelligence, given Synapse's history. But Synapse's structures, processes, and queries can be applied to other knowledge domains and data sets. **The intent of Synapse is that any data that could be represented in a spreadsheet, database, or graph database can be represented in Synapse using an appropriate data model.**

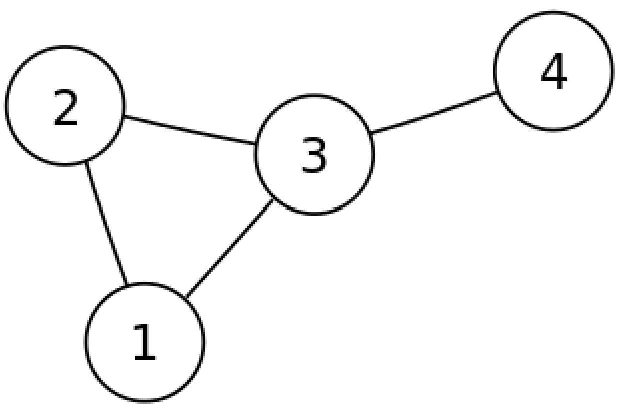
3.1.2 Graphs and Hypergraphs

To understand the power of Synapse, it helps to have some additional background. Without delving into mathematical definitions, this section introduces key concepts related to a **hypergraph,** and contrasts them with those of a **graph** or a **directed graph.** Most people should be familiar with the concept of a graph – even if not in the strict mathematical sense – or with data that can be visually represented in graph form.

Graphs

A **graph** is a mathematical structure used to model pairwise relations between objects. Graphs consist of:

- **vertices** (or **nodes**) that represent objects, and
- **edges** that connect two vertices in some type of relationship.



Edges connect exactly two nodes; they are "pairwise" or "two-dimensional". Both nodes and edges may have properties that describe their relevant features. In this sense both nodes and edges can be thought of as representational objects within the graph: nodes typically represent things ("nouns") and edges typically represent relationships ("verbs").

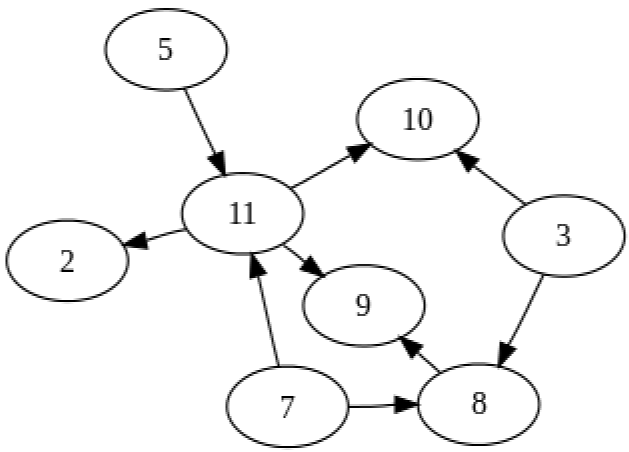
Examples

Cities and Roads. A simple example of data that can be represented by a graph are cities connected by roads. If abstracted into graph format, each city would be a vertex or node and a road connecting two cities would be an edge. Since you can travel from City A to City B or from City B to City A on the same road, the graph is **directionless** or **undirected.**

Social Networks. Another example is social networks based on "connections", such as LinkedIn. In this case, each person would be a node and the connection between two people would be an edge. In most cases, LinkedIn requires a mutual connection (you must request a connection and the other party must accept); in this sense it can be considered a directionless graph. (This is a simplification, but serves our purpose as an example.)

Directed Graphs

A **directed graph** is a graph where the edges have a direction associated with them. In other words, the relationship represented by the edge is one-way. Where an edge in an undirected graph is often represented by a straight line, an edge in a directed graph is represented by an arrow.



Examples

Cities and Roads. In our cities-and-roads example, the graph would be a directed graph if the roads were all one-way streets: in this case you can use a particular road to go from City A to City B, but not from City B to City A.

Social Networks. Social networks that support a “follows” relationship (such as Twitter) can be represented as directed graphs. Each person is still a node, but the “follows” relationship is one way – I can “follow” you, but you don’t have to follow me. If you choose to follow me, that would be a second, independent one-way edge in the opposite direction. (This is also a simplification but works for a basic illustration.)

Other Examples. Many other types of data can be represented with nodes and directed edges. For example, in information security you can represent data and relationships such as:

`malware_file --(performed DNS lookup for)--> domain`

or

`domain --(resolves to)--> ip_address`

In these examples, files, domains, and IP addresses are nodes and “performed DNS lookup for” and “resolves to” are edges (relationships). The edges are directed because a malware binary can contain programming to resolve a domain name, but a domain can’t “perform a lookup” for a malware binary; the relationship (edge) is one-way.

In addition to nodes and edges, some directed graph implementations may allow labeling or tagging of nodes and edges with additional information. These tags can act as metadata for various purposes, such as to create analytically relevant groups of objects.

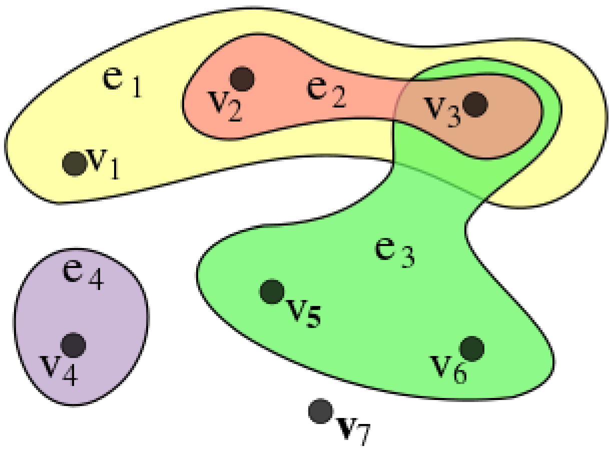
Many tools exist to visually represent various types of data in a directed graph format.

Analysis with Graphs

When working with graphs and directed graphs, analysts typically select (or lift) objects (nodes) and navigate the graph by traversing the edges (relationships) that connect those nodes. A key limitation to this approach is that all relationships (edges) between objects must be explicitly defined. You must know all of the relationships that you want to represent in advance, which makes the discovery of novel relationships among the data extremely difficult.

Hypergraphs

A **hypergraph** is a generalization of a graph in which an edge can join any number of nodes. Because an edge is no longer limited to joining exactly two nodes, edges in a hypergraph are often called **hyperedges.**



Looked at another way, the key features of a hypergraph are:

- **Everything is a node.** In a hypergraph, objects ("nouns") are still nodes, similar to a directed graph. However, relationships ("verbs", commonly represented as edges in a directed graph) may also be represented as nodes. An edge in a directed graph consists of three objects (two nodes and the edge connecting them), but in a hypergraph the same data may be represented as a single multi-dimensional node.
- **Hyperedges connect arbitrary sets of nodes.** An edge in a directed graph connects exactly two nodes. A hyperedge can connect an arbitrary number of nodes; this makes hypergraphs more challenging to visualize in a "flat" form. As in the image above, hyperedges are commonly represented as a set of disconnected nodes encircled by a boundary; the boundary represents the hyperedge "joining" the nodes into a related group. Just as there is no limit to the number of edges to or from a node in a directed graph, a node in a hypergraph can be joined by any number of hyperedges (i.e., be part of any number of "groups").

Analysis with a Synapse Hypergraph

Synapse is a specific implementation of a hypergraph model. Synapse's data store is called a **Cortex.** A Cortex is a scalable hypergraph implementation which includes key/value-based node properties and a data model that facilitates normalization.

In Synapse, all objects and most relationships are nodes (though Synapse uses what we call "lightweight" or "light" edges, similar to directed edges, in some cases). This means that most relationships in Synapse are based on nodes sharing a common property value. Instead of an FQDN being related to an IPv4 using a "resolves to" edge:

- the FQDN node is related to a DNS A record because the FQDN is a **property** of the DNS A node;
- the DNS A node is related to an IPv4 because the IPv4 is a **property** of the DNS A node.

So, in Synapse to understand the relationship between an FQDN and the IPv4 it resolves to, you navigate (pivot) from the FQDN to the DNS A node to the IPv4 node using those nodes' shared property values.

This means that in Synapse, you are not limited to navigating the data using explicitly defined edges; you primarily navigate (**pivot**) among nodes with shared property values. Synapse can readily identify these shared values, which both simplifies navigation (Synapse can "show you" the relationships; you don't need to know them in advance) and help users discover novel relationships that you may not know existed.

Synapse uses mechanisms such as **type enforcement** to ensure that properties conform to their expected values (e.g., Synapse does its best to prevent you from entering an email address where you need a URL, and that any URL you enter looks reasonably like a URL) and **property normalization** to ensure property values are represented consistently (e.g., in many cases Synapse converts string-based values to all lowercase for consistency). These methods make the data as consistent and "clean" as possible to facilitate navigation and discovery.

3.2 Data Model

Synapse's knowledge graph is built on a robust, extensible data model that can represent a broad range of data and relationships. The data model (and associated *Analytical Model*) allow both data and assertions to be represented in a structured, consistent manner. This means that instead of analysts needing to review prose reports to understand current state (and fuse those reports into still more prose to revise their assessments), analysts (and algorithms) can ask analytical questions directly of the data - and answer those questions quickly and easily.

Capturing data and analysis in a structured model abstracts away some of the subtleties and caveats that can be conveyed in prose, and finished reporting is still appropriate in many cases. But a **good** data model can represent enough information so that key objects, relationships, and assessments are well-defined, unambiguous, and self-evident upon examination.

This section provides background on the components of Synapse's data model and their use.

Tip: There are various ways to examine Synapse's data model in greater detail:

- Synapse Enterprise customers or users who have requested a Synapse demo instance and have access to the Synapse UI (Optic) can use the Data Model Explorer to view Synapse's forms and light edges and their relationships to each other. The Tag Explorer can be used to view the tags that exist in your instance of Synapse.
- Data model components such as types, forms, and properties are generated as runtime nodes ("runt nodes") when a Cortex is initialized and can be viewed as meta-objects within Synapse itself. See the *Storm Reference - Model Introspection* section for details.
- The data model is defined in the Synapse source code. The *Synapse Data Model* provides a technical reference of individual types and forms, and includes our data model deprecation policy.

3.2.1 Data Model Objects

To work effectively with Synapse and the Storm query language, you need to understand the basic elements of the Synapse data model.

Type

A **type** is the definition of a data element within the Synapse data model. A type describes what the element is and enforces how it should look, including how it should be normalized, if necessary, for both storage (including indexing) and representation (display).

Synapse's data model includes standard types such as integers and strings, but further defines a broad range of types such as globally unique identifiers (`guid`), date/time values (`time`), time intervals (`ival`), and tags (`syn:tag`).

Objects (nodes) may also be specialized types. For example, an IPv4 address (`inet:ipv4`) is its own type. An IPv4 address is stored as an integer, but the `inet:ipv4` type has additional constraints (e.g., to ensure that IPv4s created in Synapse only use integer values that fall within the allowable IPv4 address space). These constraints may be defined by a *Constructor* that specifies how a property of that type can be created (constructed) in Synapse.

Synapse uses *Type Enforcement*, *Type Normalization*, and *Type Awareness* to ensure consistency in the way data is entered, stored, and represented, and to facilitate navigation of the knowledge graph.

Type-Specific Behavior

Synapse includes optimizations for some types to improve performance and functionality. Some of these are "back end" optimizations (i.e., for indexing and storage) while some are more "front end" in terms of how users can interact with data. See *Storm Reference - Type-Specific Storm Behavior* for additional detail.

Form

A **form** is the definition of an object in the Synapse data model. A form acts as a "template" that tells you how to create a particular object (node). While the concepts of form and node are closely related, it is useful to maintain the distinction between the **template** for creating an object (a form) and an **instance** of a particular object (a node). `inet:fqdn` is a form; `inet:fqdn = vertex.link` is a node.

All forms must have a **primary property.** The primary property is the name of the form and the definition of the value to be provided for individual instances (nodes) of that form. The primary property must be defined so that it is unique across all possible instances of that form. For example, FQDNs are unique, based on the way they are defined and registered (two different organizations cannot both register the FQDN `vertex.link`). So the primary property value of an `inet:fqdn` is simply the FQDN itself.

All properties in Synapse must have a defined **type;** in many cases, a form is also its own type (for example, the form `inet:fqdn` has a type of `inet:fqdn`).

Forms may have **secondary properties** that record additional information about the form or further describe it. Secondary properties are form-specific. In most cases, secondary properties are explicitly defined for each form. If similar forms should share a subset of common properties, the properties may be defined as an **interface** that is **inherited** by those forms.

Synapse also supports a set of universal secondary properties (**universal properties**) that are valid for all forms.

Extended properties may be added to forms to store specialized or use case-specific data related to the form.

Form Namespace

Synapse uses a structured namespace for forms. Each form name consists of at least two elements separated by a colon (`:`). For example:

- `file:bytes`
- `inet:fqdn`
- `ou:org`
- `risk:threat`

The first element in the namespace represents a rough "category" for the form (i.e., `inet` for Internet-related objects). The Synapse data model is broad and extensible. The ability to group portions of the data model into related categories makes a large model easier to manage, and also allows Synapse users to focus on those portions of the model most relevant to them.

The second and / or subsequent elements in the form name define the specific "subcategory" or "thing" within the form's primary category (e.g., `inet:fqdn` represents a fully qualified domain name (FQDN) within the "Internet" (`inet`) category.

Properties have a namespace that extends the form namespace (form names are also primary properties). See *Property* and *Property Namespace* below for additional detail.

Extended Form

Synapse users can add their own **extended forms** to the data model using the *$lib.model.ext* libraries.

Note: We **strongly encourage** Synapse users who are considering extending the data model by creating custom forms to reach out to The Vertex Project first - you can readily contact us through our Slack channel. If there are gaps or missing elements in the data model, we would prefer to expand Synapse's data model for all users vs. individual users making numerous one-off customizations. If an extended form is appropriate for the use case, we can also offer feedback to help ensure the form's design is consistent with best practices.

Interface

An **interface** defines a set of secondary properties that should be present on a particular subset of forms. Instead of explicitly defining each secondary property on each form, the forms can be defined as **inheriting** a particular interface and its associated properties. This both simplifies and ensures consistency in the data model.

For example, Synapse uses several forms to represent activity occurring on a host, such as a file being added (`it:exec:file:add`) or a process being executed (`it:exec:proc`). These forms represent similar operations, so they all share a subset of secondary properties such as the time of execution (`:time`) or the file (`:exe`) or process (`:proc`) responsible for the activity. These properties are defined as an `it:host:activity` **interface** which is then declared / inherited for each form.

Interfaces can be inherited by other interfaces. For example, the `inet:proto:request` interface, which represents a client (host) requesting a network connection, inherits the `inet:host:activity` interface.

Interfaces can be used in Storm lift, filter, and pivot operations to make it easier to work with nodes of all forms that share the interface (vs. specifying each kind of node separately). See the appropriate sections of the *Storm Reference* for details.

Node

A **node** is a unique object within Synapse; they are specific instances of generic forms. Every node consists of:

- A **primary property**, represented by the form of the node plus its value (`<form> = <valu>`). All primary properties must be unique for a given form; the uniqueness of the `<form> = <valu>` pair ensures there can be only one node in Synapse that represents the domain woot.com (`inet:fqdn = woot.com`). Because this unique pair "defines" the node, the comma-separated form / value combination (`<form>,<valu>`) is also known as the node's *Ndef* (short for "node definition").
- One or more **universal properties** and an associated property value. As the name implies, universal properties apply to all nodes.
- Optional **secondary properties**. Similar to primary properties, secondary properties consist of a property name (of a specific **type**) and the property's value (`<prop> = <pval>`).
- Optional **tags**. A tag acts as a label with a particular meaning that can be applied to a node to provide context.
- Optional **extended properties** and their associated values.

Node Example

The Storm query below lifts and displays the node for the domain `www.google.com`:

```
storm> inet:fqdn=www.google.com
inet:fqdn=www.google.com
        :domain = google.com
        :host = www
        :issuffix = false
        :iszone = false
        :zone = google.com
        :_virustotal:reputation = 497
        :_virustotal:votes:harmless = 318
        :_virustotal:votes:malicious = 53
        .created = 2024/04/19 14:28:20.116
        #rep.moz.500
```

In the output above:

- `inet:fqdn = www.google.com` is the **primary property** (`<form> = <valu>`).
- `.created` is a **universal property** showing when the node was added to the Cortex.
- `:domain`, `:host`, etc. are form-specific **secondary properties** with their associated values (`<prop> = <pval>`). For readability, secondary properties (including universal properties and extended properties) are displayed as **relative properties** within the namespace of the form's primary property (e.g., `:domain` as opposed to `inet:fqdn:domain`).
- The various `:_virustotal:*` properties are **extended properties** added to the data model by the Synapse-VirusTotal Power-Up to represent specialized data provided by VirusTotal.
- `#rep.moz.500` is a **tag** indicating that `www.google.com` has been reported by web analytics company Moz as one of their top 500 most popular websites.

See *Kinds of Nodes* below for additional detail on how nodes are used to represent various objects in Synapse.

Property

Properties are the individual elements that define a **form** or (along with their values) that comprise a **node.** All properties in Synapse must have a defined **type**.

Primary Property

Every form consists of (at minimum) a **primary property:** the name of the form and the definition of the value to be provided for individual instances (nodes) of that form. All forms must be designed so that their primary property value is unique across all instances (nodes) of that form.

This uniqueness is straightforward for simple objects such as FQDNs or email addresses. Ensuring "uniqueness" for more complex nodes (such as those representing a *Relationship* or an *Event*) can be more challenging; these forms are often *GUID* forms.

Because a primary property uniquely defines a node, **it cannot be modified once the node is created.** To "change" a node's primary property value you must delete and re-create the node.

Secondary Property

A form can include optional **secondary properties** that provide additional detail about the form. Secondary properties are specific to a given form and further describe that form. A node may include secondary properties with their associated values (`<prop> = <pval>`).

Some secondary properties are based on (derived from) a node's primary property value. For example, an email address (`inet:email`) has secondary properties for both the associated FQDN (`inet:email:fqdn`) and username (`inet:email:user`). When you create the node `inet:email=info@vertex.link`, Synapse automatically sets the associated secondary property values. Any secondary properties derived from a node's primary property are read-only (just like the primary property they are based on) and cannot be changed once set.

Any secondary properties **not** based on a node's primary property are **optional.** Their values can be set if the data is available and relevant to your use case; otherwise they can remain unset. For example, an IPv4 node (`inet:ipv4`) has an optional secondary property for its associated Autonomous System (AS) number (`inet:ipv4:asn`). All optional secondary property values can be set, modified, or removed as needed.

Universal Property

Synapse defines a subset of secondary properties as **universal properties** that are applicable to all forms:

- `.created`, which is set automatically by Synapse for all nodes and whose value is the date/time that the node was created within that instance of Synapse (Cortex).
- `.seen`, which is optional for all nodes and whose value is a time interval (minimum or "first seen" and maximum or "last seen") during which the node was observed, existed, or was valid.

Extended Property

Synapse supports the addition of specialized ("extended") properties outside of Synapse's baseline data model. Extended properties may be used to represent specialized data that is relevant for specific use cases and can be added using the *$lib.model.ext* libraries.

For example, third-party vendors that provide threat intelligence or cybersecurity data may include vendor assessments, such as "risk" or "reputation" scores. These values may only be "interesting" to security researchers, and are provided by a very specific data source. Instead of adding these specialized values to the baseline data model, extended properties can be added as needed to accommodate specialized needs.

Extended properties must start with an underscore (`:_<extended_property>`) to avoid name collisions with baseline data model properties (current or future). In addition, we recommend using the name of the vendor or data source (if appropriate) as the first element in the property namespace (e.g., `:_virustotal:reputation`).

Note: We **strongly encourage** Synapse users who are considering extending the data model by creating custom properties to reach out to The Vertex Project first - you can readily contact us through our Slack channel. If there are gaps or missing elements in the data model, we would prefer to expand Synapse's data model for all users vs. individual users making numerous one-off customizations. If an extended property is appropriate for the use case, we can also offer feedback to help ensure the property's design is consistent with best practices.

Property Namespace

Properties extend the *Form Namespace*. Form names are **primary properties**, and consist of at least two elements separated by a colon (`:`).

- **Secondary properties** exist within the namespace of their primary property (form). Secondary properties are preceded by a colon (`:`) and use the colon to separate additional namespace elements, if needed.
- **Universal properties** are preceded by a period (`.`) to distinguish them from form-specific secondary properties.
- **Extended properties** are preceded by a colon and an underscore (`:_`).

For example, the secondary (both universal and form-specific) properties of `inet:fqdn` include:

- `inet:fqdn.created` (universal property)
- `inet:fqdn:zone` (secondary property)

The VirusTotal Power-Up adds extended properties to various forms, including `inet:fqdn`:

- `inet:fqdn:_virustotal:reputation`

Secondary properties (including extended and universal properties) also make up a relative namespace (set of **relative properties**) with respect to their primary property (form). The Storm query language allows (or in some cases, requires) you to reference a property using its relative property name (i.e., `:zone` vs. `inet:fqdn:zone`).

Relative properties are also used for display purposes within Synapse for visual clarity (see the *Node Example* above).

Secondary properties (including extended properties) may have their own namespace. Both primary and secondary properties use colons to separate elements of the property name. However, not all separators represent property "boundaries"; some act more as "sub-namespace" separators.

For example `file:bytes` is a primary property / form. A `file:bytes` form may include secondary properties such as `:mime:pe:imphash` and `:mime:pe:compiled`. In these examples, `:mime` and `:mime:pe` are not secondary properties, but sub-namespaces for individual MIME data types and the "PE executable" data type specifically.

Tag

Tags are annotations applied to nodes. They can be thought of as labels that provide context to the data represented by the node.

Broadly speaking, within Synapse:

- Nodes represent **things:** objects, relationships, or events. In other words, nodes typically represent observables that are verifiable and largely unchanging.
- Tags typically represent **assessments:** observations that could change if the data or the analysis of the data changes.

For example:

- An Internet domain is an "observable thing" - a domain exists, was registered through a domain registrar, and can be created as a node such as `inet:fqdn = woot.com`.
- Whether a domain has been sinkholed is an assessment. A researcher may need to evaluate data related to that domain (such as domain registration records or current and past IP resolutions) to decide whether the domain appears to be sinkholed. This assessment can be represented by applying a tag such as `cno.infra.dns.sink.holed` to the `inet:fqdn = woot.com` node.

Tags can include *Tag Timestamps* and support the addition of *Tag Properties*.

Tags are unique within the Synapse model because tags are both **nodes** and **labels applied to nodes.** The tag `cno.infra.dns.sink.holed` can be applied to another node; but the tag itself also exists as the node `syn:tag = cno.infra.dns.sink.holed`. This difference is illustrated in the example below.

Tip: Synapse does not have any pre-defined tags. Users are free to create tags that are meaningful for their analysis. See *Analytical Model* for more detail.

Tag Example

The Storm query below displays the **node** for the tag `cno.infra.dns.sink.holed`:

```
storm> syn:tag=cno.infra.dns.sink.holed
syn:tag=cno.infra.dns.sink.holed
        :base = holed
        :depth = 4
        :doc = A domain (zone) that has been sinkholed.
        :title = Sinkholed domain
        :up = cno.infra.dns.sink
        .created = 2024/04/19 14:28:20.189
```

The Storm query below displays the **tag** `cno.infra.dns.sink.holed` applied to the **node** `inet:fqdn = hugesoft.org`:

```
storm> inet:fqdn=hugesoft.org
inet:fqdn=hugesoft.org
        :domain = org
        :host = hugesoft
        :issuffix = false
        :iszone = true
        :zone = hugesoft.org
```

(continues on next page)

(continued from previous page)

```
        .created = 2024/04/19 14:28:20.212
        #cno.infra.dns.sink.holed
```

Note that a tag **applied to a node** uses the "hashtag" symbol (`#`). This is a visual cue to distinguish tags on a node from the node's secondary properties. The symbol is also used within the Storm query language syntax to reference a tag as opposed to a `syn:tag` node.

Lightweight (Light) Edge

Lightweight (light) edges are used in Synapse to provide greater flexibility and improved performance when representing certain types of relationships. A light edge is similar to an edge in a traditional directed graph; each light edge links exactly two nodes (`n1` and `n2`), and consists of:

- A **direction.** Light edge relationships only "make sense" in one direction, given the forms that they link. For example, an article can reference an indicator such as an MD5 hash, but an MD5 hash does not "reference" an article.
- A **"verb"** that represents the relationship (e.g., `refs` for "references" in the example above).

Light edges do not have properties, and you cannot apply tags to light edges - hence the "light" in light edge.

Light edges are used for performance and flexibility in certain use cases. For example:

- When the **only** information you need to record about a relationship is that it exists (that is, no properties are required to further "describe" the relationship). An example is `meta:ruleset -(contains)> meta:rule`.
- When the objects (nodes) involved in the relationship may vary. That is, either the `n1` or `n2` node (or both) may be **any** kind of node, depending on the context of the relationship. Examples include `meta:source -(seen)> *` (where a data source may "see", observe, or provide data on any `n2` object) and `* -(refs)> *` (where a variety of `n1` nodes may "reference" or contain a reference to any `n2` node).
- When the objects (nodes) to be linked do not share any properties in common (i.e., that could allow the nodes to be implicity linked via a shared property value / pivot relationship).

Synapse's source code includes some pre-defined light edges that represent The Vertex Project's conventions. While we recommend the use of these conventions, we do not enforce their use. Synapse users are free to create / define their own light edges and use them as they see fit. (Note that Synapse Power-Ups provided by The Vertex Project will create light edges according to our conventions when ingesting data.)

Tip: Light edges should not be used as a convenience to short-circuit proper data modeling using forms. Using forms and nodes (combined with Synapse's strong typing, type enforcement, and type awareness) are key to the powerful analysis and performance capabilities of a Synapse hypergraph.

3.2.2 Kinds of Forms

Synapse forms can be broadly grouped based on how their **primary properties** (`<form> = <valu>`) are formed. Recall that primary properties must be defined so that they are unique for all possible instances of that form.

Simple Form

A simple form refers to a form whose primary property is a single value. Simple forms are commonly used to represent an *Object* and are the most readily understood from a modeling perspective. The "object itself" is unique by definition, so the form's primary property value is the object. Examples of simple forms include FQDNs, IP addresses (IPv4 or IPv6), hashes, and so on.

Composite (Comp) Form

A composite (comp) form is one where the primary property is a comma-separated list of two or more elements. While no single element makes the form unique, a set of elements may be sufficiently unique to define the form. Comp forms are often (though not universally) used to represent a *Relationship*.

Fused DNS A records are an example of a comp form. A DNS A record can be uniquely defined by the combination of the domain (`inet:fqdn`) and the IP address (`inet:ipv4`) in the A record. In Synapse, an `inet:dns:a` form represents the knowledge that a given domain resolved to a specific IP at some time, or within a time window. (The universal `.seen` property captures "when" (first observed / last observed) the resolution took place.)

Guid Form

A guid (Globally Unique Identifier) form is uniquely defined by a machine-generated 128-bit number. Guids account for cases where it is impossible to uniquely define a thing based on a property or set of properties. Guids are also useful for cases where the amount of data available to create a particular object (node) may vary greatly - that is, not all properties or details are available from all data sources. A guid form gives you the flexibility (through secondary properties) to capture as much (or as little) data as is available to you.

A guid form can be considered a special case of a **simple form** where the form's value is a `<guid>`.

Forms that represent one-time events are often guid forms. Examples include host execution activity (such as `it:exec:file:add` nodes) or network activity (such as `inet:dns:request` nodes). Guid forms are also used to represent entities such as people (`ps:person`) or organizations (`ou:org`).

Note: Guid values can be arbitrary (generated ad-hoc by Synapse) or predictable / deconflictable (generated based on a specific set of inputs). See the *guid* section of *Storm Reference - Type-Specific Storm Behavior* for a more detailed discussion of this concept.

Edge (Digraph) Form

An edge (digraph) form is a specialized **composite form** where the set of values for the primary property includes at least one **ndef** ("node definition, or `<form>,<valu>` pair). An edge form is a specialized relationship form that can be used when one or both of the forms to be linked could be an arbitrary (i.e., any) form. For example, a `meta:seen` node (now replaced by a `seen` light edge) was previously used to link a `meta:source` (using the node's guid value) to an arbitrary node that was "seen" by the source (such as the domain "woot.com", using the ndef value `inet:fqdn, woot.com`).

Edge forms predate the introduction of light edges to the Synapse data model; light edges were added in order to address some of the performance overhead incurred by edge forms (i.e., it is easier and faster to create a light edge for simple relationships vs. creating an entire node simply to link two other nodes).

Edge forms may be appropriate for particular use cases, but light edges are generally preferred where possible.

Generic Form

The Synapse data model includes a number of "generic" forms that can be used to represent metadata and / or arbitrary data.

Synapse's data model can be expanded as needed, so ideally all data in Synapse would be represented using an appropriate form. However, designing a new form may require discussion, subject matter expertise, and testing against "real world" data, as well as time to implement the changes. Analysts may have a need to capture data "in the moment" without waiting for model updates. Alternatively, some data may be "one off" information that needs to be represented, but does not necessarily require its own form for a limited or unique use case.

In the above cases, generic forms may be used to capture data where a more specific form does not exist. Generic forms reside in two primary parts of the data model: `meta:*` forms and `graph:*` forms.

The `meta:rule` form is an example of a generic form. Synapse includes more specific forms to represent common detection logic such as antivirus (`it:av:sig` and `it:av:filehit`) or YARA rules (`it:app:yara:rule` and `it:app:yara:match`). Other technologies or organizations may have their own specific (and often "black box") detection logic.

A `meta:rule` form can represent an arbitrary detection rule, with a `-(matches)>` light edge used to link the rule to the "thing" (file, network traffic, etc.) that the rule fired on.

3.2.3 Kinds of Nodes

Nodes represent standard objects ("nouns") such as IP addresses, files, people, conferences, or airplanes. They can also represent more abstract objects such as industries, risks, attacks, or goals. However, in Synapse nodes can also represent relationships or specific time-based events. You can think of a node generically as a "thing" - most "things" you want to model within Synapse are nodes.

Broadly speaking, nodes can be thought of in terms of some generic categories:

Object

Nodes can represent atomic objects or entities, whether real or abstract. Entities are often (though not always) represented as a *Simple Form*. An email address (`inet:email`) is a basic example of an entity-type node / simple form:

```
storm> inet:email=kilkys@yandex.ru
inet:email=kilkys@yandex.ru
        :fqdn = yandex.ru
        :user = kilkys
        .created = 2024/04/19 14:28:20.279
```

Relationship

Nodes can represent specific **relationships** among entities. Examples include a domain resolving to an IPv4 address, a malware dropper containing or extracting another file, a company being a subsidiary of another business, or a person being a member of a group.

Relationship nodes are often represented as a *Composite (Comp) Form*. Comp forms have a primary property consisting of a comma-separated list of two or more values that uniquely define the relationship. A DNS A record (`inet:dns:a`) is a basic example of a relationship node:

```
storm> inet:dns:a=(google.com,172.217.9.142)
inet:dns:a=('google.com', '172.217.9.142')
        :fqdn = google.com
        :ipv4 = 172.217.9.142
        .created = 2024/04/19 14:28:20.325
```

Event

Nodes can represent individual time-based occurrences. The term **event** implies that an entity existed or a relationship occurred at a specific point in time. Events represent the combination of a node and a timestamp for when the node was observed. Examples of event forms include an individual login to an account, a specific DNS query, or a domain registration (whois) record captured on a specific date.

The structure of an event node may vary depending on the specific event being modeled. A "simple" event may be represented as a *Composite (Comp) Form* that combines an entity and a timestamp; for example, a domain whois record (`inet:whois:rec`) consists of the whois record and the time that record was observed or retrieved.

Other more complex events are represented as a *Guid Form* with the timestamp as one of several secondary properties on the form. A specific, individual DNS query (`inet:dns:request`) is an example of an event node:

```
storm> inet:dns:request=00000a17dbe261d10ce6ed514872bd37
inet:dns:request=00000a17dbe261d10ce6ed514872bd37
        :query = ('tcp://199.68.196.162', 'download.applemusic.itemdb.com', '1')
        :query:name = download.applemusic.itemdb.com
        :query:name:fqdn = download.applemusic.itemdb.com
        :query:type = 1
        :reply:code = 0
        :server = tcp://178.62.239.55
        :time = 2018/09/30 16:01:27.506
        .created = 2024/04/19 14:28:20.369
```

Instance Knowledge vs. Fused Knowledge

For some types of data, event nodes and relationship nodes can encode similar information but represent the difference between **instance knowledge** and **fused knowledge.**

- Event forms represent the specific point-in-time existence of an entity or occurrence of a relationship - an **instance** of that knowledge.
- Relationship forms can leverage the universal `.seen` property to set "first observed" and "last observed" times during which an entity existed or a relationship was true. This date range can be viewed as **fused** knowledge - knowledge that summarizes or "fuses" the data from many individual observations (instances) of the node over time.

Instance knowledge and fused knowledge represent differences in data granularity. Whether to create an event node or a relationship node (or both) depends on how much detail is required for your analysis. This consideration often applies to relationships that change over time, particularly those that may change frequently.

DNS records are a good example of these differences. The IP address that a domain resolves to may change infrequently (e.g., for a website hosted on a stable server) or may change quite often (e.g., where the IP is dynamically assigned or where load balancing is used).

One option to represent and track DNS A records is to create individual events every time you check the domain's current resolution (e.g., `inet:dns:request` and `inet:dns:answer` forms). This represents a very high degree of granularity as the nodes will record the exact time a domain resolved to a given IP. The nodes can also capture additional detail such as the querying client, the responding server, the response code, and so on. However, the number of such nodes could readily reach into the hundreds of millions if you create nodes for every resolution of every domain you want to track.

On the other hand, it may be sufficient to know that a domain resolved to an IP address during a given **period** of time – a "first observed" and "last observed" (`.seen`) range. A single `inet:dns:a` node can be created to show that domain `woot.com` resolved to IP address `1.2.3.4`, where the earliest observed resolution was 2014/08/06 at 13:56 and the most recently observed resolution was 2018/05/29 at 7:32. These timestamps can be extended (earlier or later) if additional data changes our observation boundaries.

This second approach loses some granularity:

- The domain is not guaranteed to have resolved to that IP **continuously** throughout the entire time period.
- Given only this node, we don't know **exactly** when the domain resolved to the IP address during that time period, except for the earliest and most recent observations.

However, this fused knowledge may be sufficient for our needs and may be preferable to creating thousands of nodes for individual DNS resolutions.

Of course, a hybrid approach is also possible, where most DNS A record data is recorded in fused `inet:dns:a` nodes but it is also possible to record high-resolution, point-in-time `inet:dns:request` and `inet:dns:answer` nodes when needed.

3.3 Analytical Model

Synapse's *Data Model* provides a structured way to record, query, and navigate "observables" - objects, relationships, and events that can be captured and are unlikely to change.

Synapse also gives analysts a structured way to record observations or assessments through the use of labels (**tags**) applied to data (nodes). Assessments represent conclusions based on the data available to you at the time. As new data becomes available, your analysis is revised. As labels on nodes, tags are flexible and can be easily added, updated, or removed when assessments change.

Tags provide immediate **context** to individual nodes. In addition, by representing both data (nodes) and assessments (tags) consistently, analysts can use Synapse to query both of these in very powerful ways.

Synapse uses the `syn:tag` form to represent tags, which is simple and straightforward. The appropriate **use** of tags to annotate data is more nuanced. You can think of tags - their structure and application - as an **analytical model** that complements and extends the power of the data model.

The annotations and assessments that are "useful" for analysis may vary widely based on the analytical discipline in question, or even the needs of individual organizations within the same discipline. For this reason, Synapse does not include any "built in" tags. Organizations are free to design and use tags and tag trees that are most useful and relevant to them.

Tip: We encourage the design and use of tags that:

- annotate assessments and conclusions that are relevant to **your** analysis.
- allow you to ask the analytical questions that are most important to **your organization.**

While many disciplines will have similar tagging needs, tags are not necessarily "one size fits all". For an example of tags/tag trees used by The Vertex Project, see our Vertex Tag Tree Overview blog.

This section discusses tags, their unique features, and their uses in more detail.

3.3.1 Tags as Nodes

Tags in Synapse are nodes (`syn:tag` nodes) in their own right. As nodes, they can be viewed directly within Synapse, making them "self-documenting" (see *Storm Reference - Model Introspection* or Optic's Tag Explorer for details on viewing and working with tags).

A tag's primary property is the name of the tag; so the tag `foo.bar` has the primary property `syn:tag = foo.bar`. The dotted notation can be used to construct tag hierarchies / tag trees to organize tags and represent varying levels of specificity. Other `syn:tag` properties allow you to record a definition for the tag and support navigation tag nodes.

This example shows the **node** for the tag `syn:tag = rep.mandiant.apt1`:

```
storm> syn:tag=rep.mandiant.apt1
syn:tag=rep.mandiant.apt1
        :base = apt1
        :depth = 2
        :doc = Indicator or activity Mandiant calls (or associates with) APT1.
        :title = APT1 (Mandiant)
        :up = rep.mandiant
        .created = 2024/04/19 14:27:26.796
```

The `syn:tag` node has the following properties:

- `.created`, which is a universal property showing when the node was added to a Cortex.
- `:title` and `:doc`, which store concise and more detailed definitions for the tag. Definitions on tag nodes help to ensure the tags are applied (and interpreted) correctly by Synapse analysts and other users.

The `:depth`, `:up`, and `:base` secondary properties help to lift and pivot across tag nodes:

- `:depth` is the "location" of the tag in a given tag tree, with the count starting from zero. A single-element tag (`syn:tag = rep`) has `:depth = 0`, while a three-element tag (`syn:tag = rep.mandiant.apt1`) has `:depth = 2`.
- `:base` is the final (rightmost) element in the tag tree.
- `:up` is the tag one "level" up in the tag tree.

Tags (`syn:tag` forms) have some specialized behaviors within Synapse with respect to how they are indexed, created, and manipulated via Storm. Most important for practical purposes is that `syn:tag` nodes are created "on the fly" when a tag is applied to another node. You do not need to create the `syn:tag` node before the tag can be used; applying the tag will automatically create the appropriate `syn:tag` node (or nodes).

See the *syn:tag* section within *Storm Reference - Type-Specific Storm Behavior* for additional detail.

3.3.2 Tags as Labels

A tag's value (`syn:tag = <valu>`) is simply a string and can be set to any user-defined alphanumeric value. Tags do not support special characters except for the underscore (`_`).

Tag strings use a dotted naming convention, with the period (`.`) used as a separator to delimit individual elements of a tag if necessary. This dotted notation supports the creation of tag hierarchies or tag trees. These trees can be used to "categorize" different types of tags (with each top-level or root tag representing a particular category). The structure can also support increasingly detailed or specific observations.

Within a tag tree, specific terms are used for the tags and their components:

- **Leaf tag:** The full tag.
- **Root tag:** The top / leftmost element in a given tag.

- **Base tag:** The bottom / rightmost element in a given tag.

For the tag `rep.microsoft.forest_blizzard`:

- `rep.microsoft.forest_blizzard` is the leaf tag (leaf).
- `rep` is the root tag (root).
- `forest_blizzard` is the base tag (base).

When you apply a tag to a node, all of the tags **above** that tag in the tag tree are automatically applied as well (and the appropriate `syn:tag` nodes are created if they do not exist). That is, when you apply the tag `rep.microsoft.forest_blizzard` to a node, Synapse automatically applies the tags `rep.microsoft` and `rep` as well. This allows you to "ask" about tags at any depth:

- `#rep.microsoft.forest_blizzard`: all things Microsoft associates with "Forest Blizzard".
- `#rep.microsoft`: all things reported by Microsoft.
- `#rep`: all things reported by any third party.

When you delete (remove) a tag from a node, the tag and all tags **below** it in the tag tree are deleted. If a node has the tag `rep.microsoft.forest_blizzard`:

- if you delete the tag `rep.microsoft.forest_blizzard` (the base tag), the tags `rep.microsoft` and `rep` will remain.
- if you delete the tag `rep` (the root or full tag) then all three tags are deleted.

Deleting a tag from a node does **not** delete the `syn:tag` node for the tag itself.

See the *syn:tag* section within *Storm Reference - Type-Specific Storm Behavior* for additional detail on tags and tag behavior.

Tag Timestamps

Synapse supports the use of optional tag **timestamps** to indicate that the assessment represented by a tag was true, relevant, or observed within the specified time window. Tag timestamps are intervals (pairs of date / time values) similar to the `.seen` universal property.

Like `.seen` properties, tag timestamps represent a time **range** and not necessarily specific instances (other than the "first known" and "last known" observations). This means that the assessment represented by the tag is not guaranteed to have been true throughout the entire date range (though depending on the meaning of the tag, that may be the case). That said, the use of timestamps allows much greater granularity in recording observations in cases where the timing of an assessment ("when" something was true or applicable) is relevant.

As an example, tag timestamps can be used to indicate when an IPv4 address was used as a TOR exit node. This knowledge can aid with both current and historical analysis of network infrastructure.

```
storm> inet:ipv4 = 185.29.8.215
inet:ipv4=185.29.8.215
        :asn = 60567
        :loc = se.ab.stockholm
        :type = unicast
        .created = 2024/04/19 14:27:26.893
        #cno.infra.anon.tor.exit = (2023/05/08 14:30:51.000, 2023/08/17 19:39:48.000)
```

The tag `cno.infra.anon.tor.exit` indicates that the IPv4 has been used as a TOR exit; the dates associated with the tag indicate the "first seen" and "last seen" times.

Tag Properties

Synapse supports the creation and use of custom **tag properties** that can provide additional context to a given tag or set of tags. Tag properties must be created programmatically before they can be used.

Once a tag property is created, it can be applied (appended) to **any** tag; they are not restricted to particular tags. Tag properties are best suited for use cases that would be applicable to **all** (or at least most) tags in your environment. A better option in many cases is the creation of **extended model properties** to represent this additional information.

For example, a third-party data vendor might provide a custom "risk" score associated with an indicator such as an FQDN. While this could be added as a custom `:risk` tag property (`#rep.somevendor:risk=80`), the `:risk` property would then be available for use with any / all tags in the environment, which may not be applicable.

Instead, an extended property can be added to the data model and the "risk" score recorded as a property on the FQDN:

`inet:fqdn:_somevendor:risk=80`

This limits the use of the vendor's "risk" score to only those forms / nodes where it is relevant, and also allows you to work with (select/lift, filter, pivot, etc.) the value the same way as any other property in the data model.

A discussion of extended model elements (forms, properties, etc.) is beyond the scope of this document. Storm libraries for working with extended model elements can be found here: *$lib.model.ext*.

3.3.3 Tags Associated with Nodes

Tags can represent observations or assessments. In some cases tags can stand on their own - the tag `cno.infra.anon.tor.exit` used to indicate that a node (such as an IPv4 address) represents anonymous network infrastructure (specifically, a TOR exit node) is straightforward. In other cases, a tag may represent or "say something" about a larger concept. The tag `rep.mandiant.apt1` means that Mandiant associates an indicator (such as a malware binary) with the threat group APT1. This provides context to the malware binary, but may create additional questions. Who or what is APT1? Where are they located? When did Mandiant first observe them?

Where a tag references a "thing" and you want to record additional information about that thing, the tag can be associated with a node (via a `:tag` secondary property). For example `risk:threat` nodes represent reporting of threat activity by a particular organization (such as Mandiant). The node's `risk:threat:tag` property can be set to `rep.mandiant.apt1`. You can then navigate from nodes that have the `rep.mandiant.apt1` tag, to the node `syn:tag=rep.mandiant.apt1`, to the `risk:threat` node with that `:tag` value to learn more about Mandiant's APT1.

Tip: An alternative method (which predates the use of `:tag` properties for certain nodes) is a "tag the tags" approach. If you wanted to record additional information "about" Mandiant's APT1, you could apply additional tags to the `syn:tag=rep.mandiant.apt1` node itself. For example, to indicate "Mandiant states APT1 is based in China", you could apply a tag such as `rep.mandiant.origin.cn` to the node `syn:tag=rep.mandiant.apt1`.

This approach is valid for some use cases and may seem easier to implement than associating tags with nodes. However, "tag the tags" typically provides less power and flexiblity (to record information and navigate associated data) in favor of convenience. The pros and cons of each approach should be weighed when making implementation decisions.

3.3.4 Tag Best Practices

The tags that you use to annotate data represent your **analytical model**. Your ability to conduct meaningful analysis depends in part on whether your analytical model is well-designed to meet your needs. The tags that work best for you may be different from those that work well for another organization.

The following recommendations should be considered when creating, maintaining, and using tags and tag trees.

Tag Trees

Tag trees generally move from "less specific" to "more specific" the deeper you go within a hierarchy. The order of elements in your hierarchy can affect the types of analysis questions you can most easily answer. The structure you create should allow you to increase specificity in a way that is meaningful to the questions you're trying to answer.

For example, let's say you are storing copies of articles from various news feeds within Synapse (i.e., as `media:news` nodes). You want to use tags to annotate the subject matter of the articles. Two possible options would be:

Tag Tree #1

```
<country>.<topic>.<subtopic>.<subtopic>:
  us.economics.trade.gdp
  us.economics.trade.deficit
  us.economics.banking.lending
  us.economics.banking.regulatory
  us.politics.elections.national
  france.politics.elections.national
  france.politics.elections.local
  china.economics.banking.lending
```

Tag Tree #2

```
<topic>.<subtopic>.<subtopic>.<country>:
  economics.trade.gdp.us
  economics.trade.deficit.us
  economics.banking.lending.us
  economics.banking.regulatory.us
  politics.elections.national.us
  politics.elections.national.france
  politics.elections.local.france
  economics.banking.lending.china
```

Neither tag tree is right or wrong; which is more suitable depends on the types of questions you want to answer. If your analysis focuses primarily on news content within a particular region, the first option (which places "country" at the root of the tree) is probably more suitable. If your analysis focuses more on global geopolitical topics, the second option is probably better. As a general rule, the analytical focus that you "care about most" should generally go at the top of the hierarchy in order to make it easier to ask those questions.

Tag Elements

Each positional element within a tag tree should have the same "category" or meaning. This makes it easier to work with portions of the tag tree in a consistent manner. For example, if you are tagging indicators of compromise with assessments related to third-party reporting, you should maintain a consistent structure:

`rep.<reporter>.<thing reported>`

In this example `rep` is a top-level namespace for third party reporting, the second element refers to the reporter, and the third element to what is being reported (threat, malware family, campaign, etc.).

Tag Precision

A tag should represent "one thing" - an atomic assessment. This makes it easier to change that specific assessment without impacting other assessments. For example, let's say you assess that an IPv4 address was used by the Vicious Wombat threat group as a C2 location for Redtree malware. It might be tempting to create a tag such as:

`cno.threat.vicious_wombat.redtree.c2`

By combining three assessments (who used the IPv4, the malware associated with the IPv4, and how the IPv4 was used) you have made it much more difficult to update the context on the IP if any one of those three assessments changes. What if you realize the IPv4 was used by Sparkling Unicorn instead? Or that the IPv4 was used for data exfiltration and not C2? Using three separate tags makes it much easier to revise your assessments if necessary:

- `cno.threat.vicious_wombat.use`
- `cno.mal.redtree`
- `cno.role.c2`

Tag Definitions

You can store both short-form and long-form definitions directly on `syn:tag` nodes using the `:title` and `:doc` properties, respectively. We recommend that you use these properties to clearly define the meaning of the tags you create within Synapse to ensure they are both applied and interpreted consistently.

Tag Depth

Tag trees can be arbitrarily deep (that is, can support an arbitrary number of tag elements). This implies that deep tag trees can potentially represent very fine-grained observations. While more detail is sometimes helpful, tag trees should reflect the level of detail that is **relevant** for your analysis, and no more. Overly-detailed tag trees can actually hamper analysis by providing too many choices for analysts.

Tags that represent analytical assertions mean that **a human analyst** typically needs to evaluate the data, make an assessment, and decide what tag (or tags) to apply to the data. If tags are overly detailed analysts may get bogged down in "analysis paralysis" - worrying about whether tag A or tag B is correct when that distinction really doesn't matter to the analysis at hand.

We recommend that tags have no more than five elements at most. As always, your specific use case may vary but this works well as general guidance.

Tag Rollout

Tagging data may represent a novel approach to analysis for many users. As analysts adjust to new workflows, it may be helpful to implement a subset of tags at first. Getting used to applying some basic tags may be easier than suddenly being asked to annotate data with a broad range of observations. As analysts get comfortable with the process, you can introduce additional tags or tag trees as appropriate.

Tag Flexibility

Tags are meant to be flexible - the ability to easily add, remove, and modify tags is a built-in aspect of Synapse. Synapse also includes tools to help move, migrate, or restructure entire tag trees (e.g., the Storm *movetag* command).

No one designs a complete, perfect tag structure from the start. It is common to design an initial tag tree and then make changes once you have tested it in practice. Your tag trees will grow over time as analysts identify new observations they want to record. Your analytical needs may change, requiring you to reorganize multiple trees.

This is fine (and expected)! **Don't be afraid to try things or change your mind.** In most cases, bulk changes and migrations can be made using Storm.

Tag Management

Any user with the appropriate permissions can create a new tag. The ability to create tags on the fly makes tags extremely flexible and convenient for analysts – they can create annotations to reflect their observations "in the moment" without the need to wait for code changes or approval cycles.

There is also some risk to this approach, particularly with large numbers of analysts, as analysts may create tags in an uncoordinated and haphazard fashion. Creating arbitrary (and potentially duplicative or contradictory) tags can work against effective analysis.

Your approach to tag creation and approval will depend on your needs and your environment. Where possible, we recommend a middle ground between "tag free-for-all" and "tightly-enforced change management". It is useful for an analyst to create a tag on demand; if they have to wait for review and approval, their observation is likely to be lost as they move on to other tasks. That said, it is also helpful to have some type of regular review process to ensure the tags are being used in a consistent manner, fit appropriately into your analytical model, and have been given clear definitions.

Official vs. "Scratch" Tags

Not all tags and tag trees need to be formally defined and approved. Many organizations define an official set of tag trees that are approved for "production" use and also define (or allow) analysts to use unofficial, personal, or "scratch" tags as needed to help with ongoing research. "Unofficial" tags should use their own namespace (for example, "int" for internal, "temp" for temporary, or "thesilence" for users' personal trees) to clearly separate them from official tags / trees but are otherwise encouraged (and highly useful).

Tag Consistency

No matter how well-designed a tag tree is, it is ineffective if the tags aren't used consistently – that is, by a majority of analysts across a majority of relevant data. It's true that 100% visibility into a given data set and 100% analyst review and annotation of that data is an unrealistic goal. However, for data and annotations that represent your **most pressing** analytical questions, you should strive for as much completeness as possible.

Looked at another way, inconsistent use of tags can result in gaps that can skew your assessment of the data. At best, this can lead to the inability to draw meaningful conclusions; at worst, to faulty analysis.

Inconsistency often occurs as both the number of analysts and the number of tags increase. The larger the team of analysts, the more difficult it is for that team to work closely and consistently together. Similarly, the more tags available to represent different assessments, the fewer tags an analyst can reasonably work with. In both cases, analysts may tend to drift towards analytical tasks that are most immediately relevant to their work or most interesting to them – thus losing sight of the collective analytical goals of the entire team.

Consider an example of tracking Internet domains that masquerade as legitimate companies for malicious purposes. If some analysts are annotating this data but others are not, your ability to answer questions about this data is skewed. Let's say Threat Cluster 12 is associated with 200 domains, and 173 of them imitate real companies, but only 42 have been annotated with "masquerade" tags (e.g., `cno.ttp.se.masq`).

If you try to use the data to answer the question "does Threat Cluster 12 consistently register domains that imitate valid companies?", your assessment is likely to be "no" (only 42 out of 200 domains have the associated tag) based on the incompletely annotated data. There are gaps in your analysis because the information to answer this question has only been partially recorded.

As the scope of analysis within Synapse increases, it is essential to recognize these gaps as a potential shortcoming that may need to be addressed. Options include:

- Establish policy around which assessments and observations (and associated tags) are essential or "required", and which are secondary ("optional" or "as time allows").
- Designate individual analysts or teams to be responsible for particular tasks and associated tags - often matching their area of expertise, such as "malware analysis".
- Leverage Synapse's tools such as triggers, cron jobs, or macros to apply tags in cases where this can be automated. Automation also helps to ensure tags are applied consistently. (See *Storm Reference - Automation* for a more detailed discussion of Synapse's automation tools.)

3.4 Views and Layers

Synapse's architecture supports the separation of data into different storage areas known as **layers.** Layers can be "stacked" to give users visibility into various combinations of data using **views.**

Views and layers are closely related to Synapse's *Fork and Merge* workflow. This section provides some high-level background on these concepts. For additional discussion of views and layers (incuding examples), see our blog on Best Practices for Views & Layers.

Tip: Views and layers are also closely tied to Synapse's **permissions** system, used to manage which users (or roles) can see (read) and edit (create, modify, delete) data. We touch on some high-level permissions concepts here, but for a full discussion see the *Synapse Admin Guide*.

3.4.1 Layers

A *Layer* is where nodes and node data are stored, where changes to Synapse's data store are made, and where **write permissions** are enforced. By default, a Synapse Cortex consists of a single layer (the **default** layer).

Layers can be used to segregate different types of data. For example:

- **Sensitive vs. non-sensitive data.** Organizations may work with data that has varying levels of sensitivity. This may include data at different "classification" levels (such as CISA's Traffic Light Protocol) or data that is subject to legal, regulatory, or business restrictions (customer data, personally identifiable information) vs. data that is considered public knowledge.
- **Vetted vs. non-vetted data.** Some organizations may publish reporting or otherwise make data in Synapse available to customers, partners, or other internal teams. This shared data is typically closely reviewed for accuracy and reliability, which may differ from "internal" data that represents ongoing research or work-in-progress.

Layers are where data is **written** in Synapse; you create nodes, modify properties, and add or remove tags in a specific layer. Changes are typically made to the **top** layer of your current **view.**

Layers are typically configured and managed by Synapse Admins. The Storm *layer* commands are used to work with layers, as are the *layer* type and its methods, and the *$lib.layer* libraries. The Optic UI provides additional GUI-based tools to view and work with layers.

3.4.2 Views

A *View* defines the data (specifically, the layer or layers) that users can see; views are where **read permissions** are enforced. In Synapse, a view consists of an ordered ("stacked") set of layers and provides visibility into the combined data from those layers. The topmost layer in the view is writeable and where any changes are made.

A default installation of Synapse consists of a single view (the **default** view) which contains the default layer; this setup may be sufficient for many use cases.

In more complex environments, you can define different views (composed of different layers) to provide different groups of analysts or other users with varying access to Synapse's data. For example:

- the SOC analyst team's view may consist of a layer with a subset of vetted data;
- the threat intel team's view may include the vetted layer and a layer for their ongoing analysis; and
- the incident response team's view may include the vetted data, the analysis data (so they can leverage it for their IR activity), and a separate layer for potentially sensitive customer data related to their investigation.

Tip: A view contains the layer(s) users can see. Visibility into a view's data is all or nothing; it is not possible to let users see "only certain nodes" or "only nodes with this tag" within a given view.

The Storm *view* commands are used to work with views, along with the *view* type and its methods and the *$lib.view* libraries. The Optic UI provides additional GUI-based tools for working with views.

3.4.3 Fork and Merge

Fork a View

Synapse includes the ability to *Fork* an existing view. When you fork a view, you create a new view with a new, empty, writeable layer on top of the layer(s) from the original (parent) view. The original layer(s) and associated data become read-only; any changes that you make in the new view (creating nodes, modifying properties, adding tags, etc.) are made to the new topmost layer.

Forked views are used for:

- Easily creating a new view that contains all of the existing layers from the original (parent) view (that is, you do not need to fully construct the new view from scratch).
- Creating a "scratch space" on top of an existing "production" view.

Forking a view allows you to make changes without affecting the underlying data. Any changes can be reviewed and either committed (**merged**) into the underlying (original) view or discarded. This makes forked views ideal for a number of purposes:

- analysts can perform exploratory research, testing an approach or hypothesis without affecting "production" data.
- junior analysts undergoing training can do their work in a space where it can be reviewed by a senior analyst for feedback before committing their work to production.
- developers can test new code or automation without affecting live data.

The Storm *view.fork* command is used to fork a view. The Optic UI includes additional GUI-based tools to work with (and fork) views, including the View Selector and View Task Bar as well as the Admin Tool.

Tip: The user who forks a view has **admin** privileges for that view (and its topmost, writeable layer). This means that users who fork a view can "do anything" within that view. However, they may be prevented from **merging** some or all of those changes, based on the write permissions associated with the underlying layer. See the *Synapse Admin Guide* for a detailed discussion of permissions, including the example provided for *Case 4 - Place guardrails around writing (creating or merging) data*.

In addition, the user who forks a view is the only one with access to the view by default. To collaborate with others within the view or to have someone review your work, you need to grant permissions to individual users (or a role or roles). See the *Synapse Admin Guide* for details on assigning permissions, or the Optic User Guide for information on granting permissions in the Optic UI.

Merge a View

Changes made in a forked view can be merged into the underlying view (in whole or in part). Alternatively, the forked view can be deleted, discarding all unmerged changes. This gives you the flexibility to:

- incrementally merge subsets of data while you continue your research;
- review and merge some (or all) of your changes when your work in the view is complete;
- optionally delete the view after merging some or all of your data;
- completely delete and discard views (and data) used for testing or that contain errors (such as if you accidentally tag 100,000 nodes or retrieve passive DNS data for IPv4 127.0.0.1).

Reviewing Changes

The Storm *diff* command can be used in both the Synapse CLI and the Optic Storm Query Bar and provides a flexible way to review some or all changes using the command's options. The **diff icon** in the Optic View Task Bar provides an alternative way to view changes.

Merging Changes

The Storm *view.merge* command can be used to merge **all** changes and optionally delete the view. In Optic, the **merge icon** in the View Task Bar will merge **all** changes and automatically delete the view.

The Storm *merge* command provides greater flexibility to view and merge data, including:

- show what **would** be merged without actually merging;
- merge all data;
- merge a subset of data based on a range of filters and selection criteria (in conjunction with the *diff* command).

The *merge* command does not delete the forked view.

Deleting a Forked View

Some merge methods can automatically or optionally delete the associated view (see above).

The Storm *view.del* command can be used to delete a forked view and its associated layer. The **delete icon** in the Optic View Task Bar will also delete a forked view. The Optic Admin Tool can also be used to manage views.

Note: Deleting a view will delete all unmerged changes in that view.

3.4.4 Best Practices

- Use dedicated layers to segregate any data that should be visible to some users but not others.
- Use views to compose the sets of data (layers) that should be visible to particular users or groups.
- We **strongly encourage** forking views for all research, analysis, and testing, no matter how trivial or incidental. This applies equally to simple (one layer / one view) Cortexes as well as those with more complex view and layer architectures. In short, **do not work directly in your production data.** It is much easier to delete a forked view (or selectively merge "good" data and discard mistakes) than it is to undo errors in production.
- Consider your organization's strategy for reviewing and merging data. Depending on how you are using forked views (training, research, testing) determine what level of review, if any, is desired (or required) before merging data. Consider whether any procedures will be enforced by agreement/consensus or the use of permissions.
- Forked views provide "scratch space" for ongoing analysis, but can also create silos of data and analysis that are inaccessible to other analysts or groups. We encourage you to develop guidance around "what" should be merged and how often in order to balance the need to more fully develop research with the desire to share data that is beneficial to other users.

  For example, analysts may enrich IOCs by pulling data from third party sources into Synapse. They may then review existing and new data to identify malware families or TTPs, or to cluster threat activity. Tags representing their assessments may be preliminary; in the meantime, the nodes created from third party data could be useful to others on their team. The *merge* command could be used to merge new or updated nodes without merging any tags while analysis continues.

3.5 Tools

3.5.1 storm

The Synapse Storm tool (commonly referred to as the **Storm CLI**) is a text-based interpreter that leverages the Storm query language (see *Storm Reference - Introduction*).

- *Connecting to a Cortex with the Storm CLI*
- *Storm CLI Basics*
- *Accessing External Commands*

Connecting to a Cortex with the Storm CLI

To access the Storm CLI you must use the `storm` module to connect to a local or remote Synapse Cortex.

Note: If you're just getting started with Synapse, you can use the Synapse Quickstart to quickly set up and connect to a local Cortex using the Storm CLI.

To connect to a local or remote Synapse Cortex using the Storm CLI, simply run the Synapse `storm` module by executing the following Python command from a terminal window, where the *<url>* parameter is the URL path to the Synapse Cortex.

```
python -m synapse.tools.storm <url>
```

The URL has the following format:

```
<scheme>://<server>:<port>/<cortex>
```

or

```
<scheme>://<user>:<password>@<server>:<port>/<cortex>
```

if authentication is used.

Example URL paths:

- `cell://vertex/storage` (default if using Synapse Quickstart)
- `tcp://synapse.woot.com:1234/cortex01`
- `ssl://synapse.woot.com:1234/cortex01`

Once connected, you will be presented with the following Storm CLI command prompt:

```
storm>
```

Storm CLI Basics

Once connected to a Synapse Cortex with the Storm CLI, you can execute any Storm queries or Storm commands directly. Detailed information on using the Storm query language to interact with data in a Synapse Cortex can be found in the *Storm Reference*.

To view a list of available **Storm commands,** type `help` from the Storm CLI prompt:

```
storm> help
```

- Detailed help for any command can be viewed by entering `-h` or `--help` after the individual command.

- For additional detail on Storm commands, see *Storm Reference - Storm Commands*.

To exit the Storm CLI, enter `!quit`:

```
storm> !quit
```

- The `!quit` command is technically an "external" (to Storm) command, so must be preceded by the bang (exclamation point) symbol.

Accessing External Commands

You can access a subset of external Synapse tools and commands from within the Storm CLI. External commands differ from native Storm commands in that they are preceded by a bang / exclamation point (!) symbol.

You can view the available **external commands** by typing `!help` from the Storm CLI prompt:

```
storm> !help
!export   - Export the results of a storm query into a nodes file.
!help     - List interpreter extended commands and display help output.
!pullfile - Download a file by sha256 and store it locally.
!pushfile - Upload a file and create a file:bytes node.
!quit     - Quit the current command line interpreter.
!runfile  - Run a local storm file.
```

Notably, the Synapse `pushfile` and `pullfile` tools (used to upload and download files from a Synapse storage *Axon*) are accessible from the Storm CLI:

```
storm> !pushfile
```

```
storm> !pullfile
```

See *pushfile* and *pullfile* for additional detail on these tools.

Help for any external command can be viewed by entering `-h` or `--help` after the command:

```
storm> !export -h
```

```
storm> !export --help
```

3.5.2 pushfile

The Synapse `pushfile` command can be used to upload files to a storage Axon (see Axon in the *Synapse Devops Guide*) and optionally create an associated *file:bytes* node in a Cortex.

Large-scale file ingest / upload is best performed using an automated feed / module / API. However, `pushfile` can be useful for uploading one-off files.

Syntax

`pushfile` is executed from an operating system command shell. The command usage is as follows:

```
usage: synapse.tools.pushfile [-h] -a AXON [-c CORTEX] [-r] [-t TAGS] filenames␣
↪[filenames ...]
```

Where:

- `AXON` is the telepath URL to a storage Axon.
- `CORTEX` is the optional path to a Cortex where a corresponding `file:bytes` node should be created.

  - **Note:** while this is an optional parameter, it doesn't make much sense to store a file in an Axon that can't be referenced from within a Cortex.
- `TAGS` is an optional list of tags to be applied to the `file:bytes` node created in the Cortex.
  - `-t` takes a comma separated list of tags.
  - The tag should be specified by name only (i.e., without the # character).
- `-r` recursively finds all files when a glob pattern is used for a file name.
- `filenames` is one or more names (with optional paths), or glob patterns, to the local file(s) to be uploaded.
  - If multiple file names are specified, any tag provided with the `-t` option will be added to **each** uploaded file.

Example

Upload the file `myreport.pdf` to the specified Axon, create a `file:bytes` node in the specified Cortex, and tag the `file:bytes` node with the tag `#sometag` (replace the Axon and Cortex path below with the path to your Cortex. Note that the command is wrapped for readability):

```
python -m synapse.tools.pushfile -a tcp://axon.vertex.link:5555/axon00
  -c tcp://cortex.vertex.link:4444/cortex00 -t sometag /home/user/reports/myreport.pdf
```

Executing the command will result in various status messages (lines are wrapped for readability):

```
2019-07-03 11:46:30,567 [INFO] log level set to DEBUG
  [common.py:setlogging:MainThread:MainProcess]
2019-07-03 11:46:30,568 [DEBUG] Using selector: EpollSelector
  [selector_events.py:__init__:MainThread:MainProcess]

adding tags: ['sometag']
Uploaded [myreport.pdf] to axon
file: myreport.pdf (2606351) added to core
  (sha256:229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0) as myreport.
↪pdf
```

- `adding tags: ['sometag']` indicates the tag `#sometag` was applied to the `file:bytes` node.
- `Uploaded [myreport.pdf] to axon` indicates the file was successfully uploaded to the storage Axon.
- `file: myreport.pdf (2606351) added to core (sha256:229cdde4...5cd40ab0) as myreport.pdf` indicates the `file:bytes` node was created in the Cortex.
  - The message gives the new node's primary property value (`sha256:229cdde419ba9549023de39c6a0ca8af74b45fade2d7` and also notes the `:name` secondary property value assigned to the node (`myreport.pdf`).
  - `pushfile` sets the `file:bytes:name` property to the base name of the local file being uploaded.

If a given file already exists in the Axon (deconflicted based on the file's SHA256 hash), `pushfile` will not re-upload the file. However, the command will still process any other options, including:

- creating the `file:bytes` node in the Cortex if it does not already exist.
- applying any specified tag.
- setting (or overwriting) the `:name` property on any existing `file:bytes` node with the base name of the local file specified.

For example (lines wrapped for readability):

```
python -m synapse.tools.pushfile -a tcp://axon.vertex.link:5555/axon00
  -c tcp://cortex.vertex.link:4444/cortex00 -t anothertag,athirdtag
    /home/user/reports/anotherreport.pdf

2019-07-03 11:59:03,366 [INFO] log level set to DEBUG
  [common.py:setlogging:MainThread:MainProcess]
2019-07-03 11:59:03,367 [DEBUG] Using selector: EpollSelector
  [selector_events.py:__init__:MainThread:MainProcess]

adding tags: ['anothertag'. 'athirdtag']
Axon already had [anotherreport.pdf]
file: anotherreport.pdf (2606351) added to core
  (sha256:229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0)
    as anotherreport.pdf
```

Note the status indicating the Axon already had the specified file. Similarly, the status noting the `file:bytes` node was added to the Cortex lists the same SHA256 hash as our first upload (i.e., `anotherreport.pdf` has the same SHA256 hash as `myreport.pdf`) and indicates the `:name` property has been updated (as `anotherreport.pdf`).

The `file:bytes` node for the uploaded report can now be viewed in the specified Cortex by lifting (see *Storm Reference - Lifting*) the file using the SHA256 / primary property value from the `pushfile` status output:

```
file:bytes=sha256:229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0

file:bytes=sha256:229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0
    .created = 2019/07/03 18:46:40.542
    :md5 = 23a14d3a4508628e7e09a4c4868dfb17
    :mime = ??
    :name = anotherrepport.pdf
    :sha1 = 99b6b984988581cae681f65b92198ed77609bd11
    :sha256 = 229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0
    :size = 2606351
    #anothertag
    #athirdtag
    #sometag
complete. 1 nodes in 3 ms (333/sec).
```

Viewing the node's properties, we see that Synapse has set the `:name` property and has calculated and set the MD5, SHA1, and SHA256 hash secondary property values, as well as the file's size in bytes. Similarly the two tags from our two example `pushfile` commands have been added to the node.

Alternatively, a glob pattern could be used to upload all PDF files in a given directory:

```
python -m synapse.tools.pushfile -a tcp://axon.vertex.link:5555/axon00
  -c tcp://cortex.vertex.link:4444/cortex00 -t anothertag,athirdtag
    /home/user/reports/*.pdf
```

3.5.3 pullfile

The Synapse `pullfile` command can be used to retrieve (download) one or more files from a storage Axon (see Axon in the *Synapse Devops Guide*).

Syntax

`pullfile` is executed from an operating system command shell. The command usage is as follows:

```
usage: synapse.tools.pullfile [-h] -a AXON [-o OUTPUT] [-l HASHES]
```

Where:

- `AXON` is the telepath URL to a storage Axon.
- `OUTPUT` is the optional directory path where the downloaded file(s) should be written.
  - If no option is specified, the file(s) will be written to the current working directory.
  - It is not possible to specify multiple `-o` options with a single `pullfile` command (i.e., a different `-o` option with each `-l HASH`, for example). If multiple `-o` options are specified, the last `OUTPUT` path specified will be used.
  - Files saved locally are named using their SHA256 hash value.
- `HASHES` is the SHA256 hash(es) of the file(s) to be retrieved.
  - Multiple hashes can be specified, but each must be listed with its own `-l` option (i.e., `-l HASH_0 -l HASH_1 ... -l HASH_n`).

Example

Download the two files with the specified SHA256 hashes from the specified Axon to the local `/home/user/Documents` directory (replace the Axon path below with the path to your Axon. Note that the command is wrapped for readability):

```
python -m synapse.tools.pullfile -a tcp://axon.vertex.link:5555/axon00
  -o /home/user/Documents
  -l 229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0
  -l 52c672f45adacca4878461c1bdd5800af8518e675819a0bdcd5c64a72075a478
```

Executing the command will result in various status messages showing the query and successful retrieval of the file(s):

```
Fetching 229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0 to file
Fetched 229cdde419ba9549023de39c6a0ca8af74b45fade2d7a22cdc4105a75cd40ab0 to file
Fetching 52c672f45adacca4878461c1bdd5800af8518e675819a0bdcd5c64a72075a478 to file
Fetched 52c672f45adacca4878461c1bdd5800af8518e675819a0bdcd5c64a72075a478 to file
```

3.5.4 feed

The Synapse `feed` tool is a way to ingest data exported from one Cortex into another Cortex. Users should be familiar with both the Synapse data model (*Data Model Objects* et al.) as well as Synapse concepts such as packed nodes in order to use and understand the `feed` tool effectively.

Syntax

The `feed` tool is executed from an operating system command shell. The command usage is as follows (line is wrapped for readability):

```
usage: synapse.tools.feed [-h] (--cortex CORTEX | --test) [--debug] [--format FORMAT] [--
↪modules MODULES]
  [--chunksize CHUNKSIZE] [--offset OFFSET] [files ...]
```

Where: `--h` displays detailed help and these command line options - `CORTEX` specifies the telapth URL to the Cortex where the data should be ingested.

- `--test` means to perform the ingest against a temporary, local Cortex instead of a live cortex, for testing or validation
  - When using a temporary Cortex, you do not need to provide a path.
- `--debug` specifies to drop into an interactive prompt to inspect the state of the Cortex post-ingest.
- `FORMAT` specifies the format of the input files.
  - Currently, only the value "syn.nodes" is supported. This is also the default value.
- `MODULES` specifies a path to a Synapse CoreModule class that will be loaded into the temporary Cortex.
  - This option has no effect if the `--test` option is not specified
- `CHUNKSIZE` specifies how many lines or chunks of data to read at a time from the given files.
  - Defaults to 1000 if not specified
- `OFFSET` specifies how many chunks of data to skip over (starting at the beginning)
- `files` is a series of file paths containing data to load into the Cortex (or temporary Cortex)
  - Every file must be either json-serialized data, msgpack-serialized data, yaml-serialized data, or a json lines file. The files do not have to all be of the same type.

Ingest Examples - Overview

The `feed` tool

Ingest Example 1

This example demonstrates loading a set of nodes via the `feed` tool with the "syn.nodes" format option. The nodes are of a variety of types, and are encoded in a json lines (jsonl) format.

JSONL File:

The jsonl file (`testnodes.jsonl`) contains a list of nodes in their packed form. Each line in the file corresponds to a single node, with all of the properties, tags, and nodedata on the node encoded in a json friendly format.

```
[["it:reveng:function", "9710579930d831abd88acff1f2ecd04f"], {"iden":
↪"508204ebc73709faa161ba8c111aec323f63a78a84495694f317feb067f41802", "tags": {"my":␣
↪[null, null], "my.cool": [null, null], "my.cool.tag": [null, null]}, "props": {".
↪created": 1625069466909, "description": "An example function"},   "tagprops": {},
↪"nodedata": {}, "path": {}}]
[["inet:ipv4", 386412289], {"iden":
↪"d6270ca2dc592cd0e8edf8c73000f80b63df4bcd601c9a631d8c68666fdda5ae", "tags": {"my":␣
↪[null, null], "my.cool": [null, null], "my.cool.tag": [null, null]}, "props": {".
↪created": 1625069584577, "type": "unicast"}, "tagprops": {}, "nodedata": {}, "path": {}
↪}]
[["inet:url", "https://synapse.docs.vertex.link/en/latest/synapse/userguide.html
↪#userguide"], {"iden":
↪"dba0a280fc1f8cf317dffa137df0e1761b6f94cacbf56523809d4f17d8263840", "tags": {"my":␣
↪[null, null], "my.cool": [null, null], "my.cool.tag": [null, null]}, "props": {".
↪created": 1625069758843, "proto": "https", "path": "/en/latest/synapse/userguide.html
↪#userguide", "params": "", "fqdn": "synapse.docs.vertex.link", "port": 443, "base":
↪"https://synapse.docs.vertex.link/en/latest/synapse/userguide.html#userguide"},
↪"tagprops": {}, "nodedata": {}, "path": {}}]
[["file:bytes", "sha256:ffd19426d3f020996c482255b92a547a2f63afcfc11b45a98fb3fb5be69dd75c
↪"], {"iden": "137fd16d2caab221e7580be63c149f83a11dd11f10f078d9f582fedef9b57ad5", "tags
↪": {"my": [null, null], "my.cool": [null, null], "my.cool.tag": [null, null]}, "props
↪": {".created": 1625070470041, "sha256":
↪"ffd19426d3f020996c482255b92a547a2f63afcfc11b45a98fb3fb5be69dd75c", "md5":
↪"be1bb5ab2057d69fb6d0a9d0684168fe", "sha1": "57d13f1fa2322058dc80e5d6d768546b47238fcd",
↪ "size": 16}, "tagprops": {}, "nodedata": {}, "path": {}}]
```


Verifying the Data:

Typically, users will want to double check the data they have before loading it into a production Cortex. The `feed` tool allows us to perform an ingest our of nodes file against an empty, ephemeral Cortex, so that we can check what nodes get created before adding them to a production Cortex. To load `testnodes.jsonl` into an ephemeral Cortex and drop into a prompt to explore the ingested nodes, run:

```
python -m synapse.tools.feed --test --debug testnodes.jsonl
```

Assuming the command completed with no errors, we should now have a `cmdr` prompt connected to our test Cortex:

```
cli>
```

From which we can issue Storm commands to interact with and validate the nodes that were just ingested. For example:

```
cli> storm #my.cool.tag

it:reveng:function=9710579930d831abd88acff1f2ecd04f
        .created = 2021/06/30 19:46:31.810
        :description = An example function
        #my.cool.tag
inet:ipv4=23.8.47.1
        .created = 2021/06/30 19:46:31.810
        :type = unicast
        #my.cool.tag
inet:url=https://synapse.docs.vertex.link/en/latest/synapse/userguide.html#userguide
        .created = 2021/06/30 19:46:31.810
        :base = https://synapse.docs.vertex.link/en/latest/synapse/userguide.html
```

(continues on next page)

(continued from previous page)

```
↪#userguide
        :fqdn = synapse.docs.vertex.link
        :params =
        :path = /en/latest/synapse/userguide.html#userguide
        :port = 443
        :proto = https
        #my.cool.tag
file:bytes=sha256:ffd19426d3f020996c482255b92a547a2f63afcfc11b45a98fb3fb5be69dd75c
        .created = 2021/06/30 19:46:31.810
        :md5 = be1bb5ab2057d69fb6d0a9d0684168fe
        :sha1 = 57d13f1fa2322058dc80e5d6d768546b47238fcd
        :sha256 = ffd19426d3f020996c482255b92a547a2f63afcfc11b45a98fb3fb5be69dd75c
        :size = 16
        #my.cool.tag
complete. 4 nodes in 16 ms (250/sec).
```

Loading the Data:

Once we've inspected and verified the data is acceptable for loading, we can point the `feed` tool to the Cortex we want to load the nodes into, and the same nodes should be added.

```
python -m synapse.tools.feed --cortex "aha://cortex..." testnodes.jsonl
```

However, once we've inspected the data, let's say that the it:reveng:function and inet:ipv4 nodes are not allowed in the production Cortex, but the inet:url and file:bytes are. We can skip these two nodes by using a combination of the `chunksize` and `offset` parameters:

```
python -m synapse.tools.feed --cortex "aha://cortex..." testnodes.jsonl --chunksize 1 --
↪offset 1
```

With the `chunksize` parameter signifying that the `feed` tool should read two lines at a time from the file and process those before reading the next line, and the `offset` parameter meaning the `feed` tool should skip all lines before and including line 1 (so lines 1 and 0) when attempting to add nodes, and only add nodes once it's read in lines 2 and beyond.

3.5.5 csvtool

The Synapse `csvtool` command can be used to ingest structured data from a comma-separated values (CSV) file to create nodes in a Cortex. `csvtool` is useful for bulk-loading CSV-formatted data without the need to develop custom ingest code. (For other data formats such as JSON, yaml, or msgpack, see *feed*.)

The `--export` option can be used to export a set of data from a Cortex into a CSV file.

Storm queries are used both to ingest and export data using `csvtool`. Users should be familiar with the Storm query language (*Storm Reference - Introduction* et al.) and the Synapse data model (*Data Model Objects* et al.) in order to use `csvtool` effectively.

The Storm syntax used with `csvtool` makes use of a few more advanced Storm concepts such as variables, methods, libraries, and some programming flow control concepts (e.g., for loops and switch statements). However, the examples below should be fairly self-explanatory. In other words, users do **not** need to understand in detail how those concepts work in order to create basic `stormfile` queries and start loading data using `csvtool`.

That said, the set of advanced Storm concepts and features can be fully leveraged within a `stormfile` to perform complex data ingest. Interested users are encouraged to refer to the appropriate sections of the Storm reference documents for a more detailed discussion of those concepts, which may be useful for creating more complex `stormfile` queries (or Storm queries in general).

- *Storm Reference - Subqueries*
- *Storm Reference - Advanced - Variables*
- *Storm Reference - Advanced - Methods*
- *Storm Reference - Advanced - Control Flow*
- *Storm Libraries*
- *Storm Types*

Syntax

`csvtool` is executed from an operating system command shell. The command usage is as follows (line is wrapped for readability):

```
usage: synapse.tools.csvtool [-h] [--logfile LOGFILE] [--csv-header] [--cli] [--debug]
  (--cortex CORTEX | --test) [--export] stormfile csvfiles [csvfiles ...]
```

Where:

- `-h` displays detailed help and examples.
- `LOGFILE` is the optional name / path to log Storm events associated with running the `csvtool` command as a JSONL file. Messages are appended to this file when they are written to them.
- `--csv-header` is an option that indicates the first row in the CSV file is a header row and should be skipped for purposes of parsing and node creation.
- `--cli` opens a `cmdr` command prompt after `csvtool` exits.
  - The command prompt will be connected to the Cortex specified by the `--cortex CORTEX` or `--test` option.
- `--debug` will send verbose output to `stdout` during execution.
- `CORTEX` specifies the telepath URL to the Cortex where the data should be ingested.
- `--test` specifies the data should be loaded into a temporary local Cortex (i.e., for testing / validation).
  - When using a temporary Cortex, you do not need to provide a path.
- `--export` is used to extract data from the specified Cortex into a CSV file.
- `stormfile` is the name / path to a file containing a Storm query that tells Synapse how to ingest the CSV data (or how to lift and export data if the `--export` option is used).
- `csvfiles` is the name / path to one or more CSV files containing the data to be ingested (or the name/path where the CSV output should be written if the `--export` option is used).
  - If multiple `csvfiles` are listed for ingest, they are all processed with the specified `stormfile`.
  - Only a single `csvfile` can be specified for output with `--export`.

Note: The same events are output by both `--logfile` and `--debug`; one is written to file and the other is written to `stdout`.

---

help

The detailed help (-h) output for csvtool is shown below (lines are wrapped for readability).

```
python -m synapse.tools.csvtool -h

usage: synapse.tools.csvtool [-h] [--logfile LOGFILE] [--csv-header] [--cli] [--debug]
  (--cortex CORTEX | --test) [--export] stormfile csvfiles [csvfiles ...]

Command line tool for ingesting csv files into a cortex

The storm file is run with the CSV rows specified in the variable "rows" so most storm␣
↪files
  will use a variable based for loop to create edit nodes.  For example:

for ($fqdn, $ipv4, $tag) in $rows {
    [ inet:dns:a=($fqdn, $ipv4) +#$tag ]
}

More advanced uses may include switch cases to provide different logic based on a
  column value.

for ($type, $valu, $info) in $rows {

    switch $type {
        fqdn: {
            [ inet:fqdn=$valu ]
        }

        "person name": {
            [ ps:name=$valu ]
        }

        *: {
            // default case...
        }
    }

    switch $info {
        "known malware": { [+#cno.mal] }
    }
}

positional arguments:

stormfile             A STORM script describing how to create nodes
                      from rows.
csvfiles              CSV files to load.

optional arguments:
-h, --help            show this help message and exit
--logfile LOGFILE     Set a log file to get JSON lines from the
                      server events.
```

(continues on next page)

(continued from previous page)

```
--csv-header          Skip the first line from each CSV file.
--cli                 Drop into a cli session after loading data.
--debug               Enable verbose debug output.
--cortex CORTEX, -c CORTEX
                      The telepath URL for the cortex ( or alias
                      from ~/.syn/aliases ).
--test, -t            Perform a local CSV ingest against a temporary
                      cortex.
--export              Export CSV data to file from storm using
                      $lib.csv.emit(...) events.
```

Ingest Examples - Overview

The key components for using the `csvtool` command are the CSV file itself (`csvfile`) and the file containing the Storm query (`stormfile`) used to ingest the data.

The `stormfile` contains a Storm query to describe how the data from the CSV file(s) should be used to create nodes in a Cortex, including optionally setting properties and / or adding tags.

Note: When ingesting large sets of CSV-formatted data where the data has not been vetted, it may be useful to use the *"Try" Operator* operator instead of the equivalent (=) operator within the Storm syntax in the `stormfile` used to create nodes. When using the try operator (?=), Storm will process what it can, creating nodes from "well-formatted" data and simply skipping rows that may contain bad data. In contrast, using the equivalent operator (=) will result in Storm throwing an error and halting processing if bad data is encountered.

Ingest Example 1

This example demonstrates loading a structured set of data to create nodes of a single form (in this case, DNS A records) and set secondary properties (in this case, the `.seen` universal property).

CSV File:

A CSV file (`testfile.csv`) contains a list of domains, the IP addresses the domains have resolved to, and the first and last observed times for the resolution, as represented by the example header and row data below:

```
domain,IP,first,last
woot.com,1.2.3.4,2018/04/18 13:12:47,2018/06/23 09:45:12
hurr.net,5.6.7.8,2018/10/03 00:47:29,2018/10/04 18:26:06
derp.org,4.4.4.4,2019/06/09 09:00:18,2019/07/03 15:07:52
```

Note: Because the file contains a header row, we need to use the `--csv-header` option to tell `csvtool` to skip the first row when ingesting data.

We want to load the data in the CSV file into a Cortex as a set of DNS A records (`inet:dns:a` nodes) with the first and last dates represented as the `.seen` universal property.

Stormfile:

Storm references the set of rows in the CSV file by the *$rows* built-in variable. We need to define a set of variables (see *Storm Reference - Advanced - Variables*) to represent each field in a row (i.e., each column in the CSV file) and tell Storm to iterate over each row using a *For Loop*. For example:

```
for ($fqdn, $ipv4, $first, $last) in $rows
```

This assigns the variable `$fqdn` to the first column (i.e., the one containing `woot.com`), `$ipv4` to the second column, and so on, and sets up the "for" loop.

We then need a Storm query that tells the "for" loop what to do with each row - that is, how to create the DNS A records from each row in the CSV file:

```
[ inet:dns:a = ( $fqdn, $ipv4 ) .seen=( $first, $last ) ]
```

We combine these elements to create our `stormfile`, as follows:

```
for ($fqdn, $ipv4, $first, $last) in $rows {

    [ inet:dns:a = ( $fqdn, $ipv4 ) .seen=( $first, $last ) ]

}
```

Testing the Ingest:

Typically, users will want to test that their `stormfile` loads and formats the data correctly by first ingesting the data into a local test cortex (`--test`) before loading the data into a production Cortex. This is typically done using either the `--debug` or `--logfile` option to check for errors and reviewing the loaded data (via `--cli`).

Testing the data will highlight common errors such as:

- Invalid Storm syntax in the `stormfile`.
- Data in the CSV file that does not pass *Type* validation on node creation (i.e., bad or incorrect data, such as an IP address in an FQDN column).

We can attempt to load our data into a test Cortex using the following command (line is wrapped for readability):

```
python -m synapse.tools.csvtool --logfile mylog.json --csv-header --cli --test
  stormfile testfile.csv
```

Assuming the command executed with no errors, we should have a `cmdr` CLI prompt for our local test Cortex:

```
cli>
```

We can now issue Storm commands to interact with and validate the data (i.e., did `csvtool` create the expected number of nodes, were the properties set correctly, etc.)

For example:

```
cli> storm inet:dns:a

inet:dns:a=('hurr.net', '5.6.7.8')
     .created = 2019/07/03 22:25:43.966
    .seen = ('2018/10/03 00:47:29.000', '2018/10/04 18:26:06.000')
     :fqdn = hurr.net
     :ipv4 = 5.6.7.8
inet:dns:a=('derp.org', '4.4.4.4')
     .created = 2019/07/03 22:25:43.968
    .seen = ('2019/06/09 09:00:18.000', '2019/07/03 15:07:52.000')
    :fqdn = derp.org
```

(continues on next page)

(continued from previous page)

```
    :ipv4 = 4.4.4.4
inet:dns:a=('woot.com', '1.2.3.4')
    .created = 2019/07/03 22:25:43.962
    .seen = ('2018/04/18 13:12:47.000', '2018/06/23 09:45:12.000')
    :fqdn = woot.com
    :ipv4 = 1.2.3.4
complete. 3 nodes in 12 ms (250/sec).
```

Loading the Data:

Once we have validated that our data has loaded correctly, we can modify our `csvtool` command to load the data into a live Cortex (replace the Cortex path below with the path to your Cortex; line is wrapped for readability):

```
python -m synapse.tools.csvtool --logfile mylog.json --csv-header
  --cortex tcp://cortex.vertex.link:4444/cortex00 stormfile testfile.csv
```

Ingest Example 2

This example demonstrates loading a more complex set of data to create nodes of multiple types, apply a single tag to all nodes, and apply custom tags to only some nodes based on additional criteria.

CSV File:

A CSV file (`testfile.csv`) contains a set of malicious indicators, listed by type and the indicator value, as represented by the example header and row data below:

```
Indicator type,Indicator,Description
URL,http://search.webstie.net/,
FileHash-SHA256,b214c7a127cb669a523791806353da5c5c04832f123a0a6df118642eee1632a3,
FileHash-SHA256,b20327c03703ebad191c0ba025a3f26494ff12c5908749e33e71589ae1e1f6b3,
FileHash-SHA256,7fd526e1a190c10c060bac21de17d2c90eb2985633c9ab74020a2b78acd8a4c8,
FileHash-SHA256,b4e3b2a1f1e343d14af8d812d4a29440940b99aaf145b5699dfe277b5bfb8405,
hostname,dns.domain-resolve.org,
hostname,search.webstie.net,
```

Note that while the CSV file contains a header field titled "Description", that field in this particular file contains no data.

Let's say that in addition to the raw indicators, we know that the indicators came from a blog post describing the activity of the Vicious Wombat threat group, and that the SHA256 hashes are samples of the UMPTYSCRUNCH malware family. To provide additional context for the data in our Cortex, we want to:

- Tag all of the indicators as associated with Vicious Wombat (`#cno.threat.viciouswombat`).
- Tag all of the SHA256 hashes as associated with UMPTYSCRUNCH malware (`#cno.mal.umptyscrunch`).

Stormfile:

Similar to our first example, we need to define a set of variables to represent each column (field) for each row and set up the "for" loop:

```
for ($type, $value, $desc) in $rows
```

In this case, the rows contain different types of data that will be used to create different nodes (forms). The `Indicator type` column (`$type`) tells us what type of data is available and what type of node we should create. We can use a "switch" statement to tell Storm how to handle each type of data (i.e., each value in the `$type` field). Since we know the SHA256 hashes refer to UMPTYSCRUNCH malware samples, we want to add tags to those nodes:

```
switch $type {

    URL: {
        [ inet:url = $value ]
    }

    FileHash-SHA256: {
        [ hash:sha256 = $value +#cno.mal.umptyscrunch ]
    }

    hostname: {
        [ inet:fqdn = $value ]
    }
}
```

Finally, because we know all of the indicators are associated with the Vicious Wombat threat group, we want to add a tag to all of the indicators. We can add that after the "switch" statement:

```
[ +#cno.threat.viciouswombat ]
```

So our full `stormfile` script looks like this:

```
for ($type, $value, $desc) in $rows {

    switch $type {

        URL: {
            [ inet:url = $value ]
        }

        FileHash-SHA256: {
            [ hash:sha256 = $value +#cno.mal.umptyscrunch ]
        }

        hostname: {
            [ inet:fqdn = $value ]
        }
    }

    [ +#cno.threat.viciouswombat ]
}
```

Testing the Ingest:

We can now test our ingest by loading the data into a test Cortex (line is wrapped for readability):

```
python -m synapse.tools.csvtool --logfile mylog.json --csv-header --cli --test
  stormfile testfile.csv
```

From the `cmdr` CLI, we can now query the data to make sure the nodes were created and the tags applied correctly. For example:

Check that two `inet:fqdn` nodes were created and given the `#cno.threat.viciouswombat` tag:

```
cli> storm inet:fqdn#cno

inet:fqdn=search.webstie.net
    .created = 2019/07/05 14:49:20.110
    :domain = webstie.net
    :host = search
    :issuffix = False
    :iszone = False
    :zone = webstie.net
    #cno.threat.viciouswombat
inet:fqdn=dns.domain-resolve.org
    .created = 2019/07/05 14:49:20.117
    :domain = domain-resolve.org
    :host = dns
    :issuffix = False
    :iszone = False
    :zone = domain-resolve.org
    #cno.threat.viciouswombat
complete. 2 nodes in 14 ms (142/sec).
```

Check that four `hash:sha256` nodes were created and given both the Vicious Wombat and the UMPTYSCRUNCH tags:

```
cli> storm hash:sha256

hash:sha256=7fd526e1a190c10c060bac21de17d2c90eb2985633c9ab74020a2b78acd8a4c8
    .created = 2019/07/05 14:49:20.115
    #cno.mal.umptyscrunch
    #cno.threat.viciouswombat
hash:sha256=b20327c03703ebad191c0ba025a3f26494ff12c5908749e33e71589ae1e1f6b3
    .created = 2019/07/05 14:49:20.115
    #cno.mal.umptyscrunch
    #cno.threat.viciouswombat
hash:sha256=b214c7a127cb669a523791806353da5c5c04832f123a0a6df118642eee1632a3
    .created = 2019/07/05 14:49:20.113
    #cno.mal.umptyscrunch
    #cno.threat.viciouswombat
hash:sha256=b4e3b2a1f1e343d14af8d812d4a29440940b99aaf145b5699dfe277b5bfb8405
    .created = 2019/07/05 14:49:20.116
    #cno.mal.umptyscrunch
    #cno.threat.viciouswombat
complete. 4 nodes in 3 ms (1333/sec).
```

Loading the Data:

Once the data has been validated, we can load it into our live Cortex (replace the Cortex path below with the path to your Cortex; line is wrapped for readability):

```
python -m synapse.tools.csvtool --logfile mylog.json --csv-header
  --cortex tcp://cortex.vertex.link:4444/cortex00 stormfile testfile.csv
```

Export Examples - Overview

The `--export` option allows you to export a set of data from a Cortex into a CSV file.

When `--export` is used:

- `stormfile` contains:
  - the Storm query that specifies the data to be exported; and
  - a statement telling Storm how to format and generate the rows of the CSV file.
- `csvfile` is the location where the data should be written.

The Storm `$lib.csv` library includes functions for working with CSV files. The `$lib.csv.emit()` function will emit CSV rows; the parameters passed to the function define the data that should be included in each row.

`$lib.csv.emit()` will create one row for each node that it processes (i.e., each node in the Storm "pipeline" that passes through the `$lib.csv.emit()` command), as determined by the preceding Storm query.

Export Example 1

For this example, we will export the data we imported in *Ingest Example 2*. For this simple example, we want to export the set of malicious indicators associated with the Vicious Wombat threat group.

Stormfile:

To lift all the indicators associated with Vicious Wombat, we can use the following Storm query:

```
#cno.threat.viciouswombat
```

We then need to tell `$lib.csv.emit()` how to format our exported data. We want to list the indicator type (its form) and the indicator itself (the node's primary property value).

While this seems pretty straightforward, there are two considerations:

- Given our example above, we have multiple node types to export (`inet:url`, `hash:sha256`, `inet:fqdn`).
- While we can reference any secondary property directly using its relative property name (i.e., `:zone` for `inet:fqdn:zone`), referencing the primary property value is a bit trickier, as is referencing the form of the node.

$node is a built-in Storm variable that represents the **current node** passing through the Storm pipeline. `$node` supports a number of methods (*Storm Reference - Advanced - Methods*) that allow Storm to access various attributes of the current node. In this case:

- The *$node.form()* method will access (return) the current node's form.
- The *$node.value()* method will access (return) the current node's primary property value.

This means we can tell `$lib.csv.emit()` to create a CSV file with a list of indicators as follows:

```
$lib.csv.emit($node.form(), $node.value())
```

So our overall `stormfile` to lift and export all of the Vicious Wombat indicators is relatively simple:

```
#cno.threat.viciouswombat
$lib.csv.emit($node.form(), $node.value())
```

Exporting the Data:

We can now test our export of the data we ingested in *Ingest Example 2* (replace the Cortex path below with the path to your Cortex; line is wrapped for readability):

```
python -m synapse.tools.csvtool --debug --export
  --cortex tcp://cortex.vertex.link:4444/cortex00 stormfile export.csv
```

If we view the contents of `export.csv`, we should see our list of indicators:

```
inet:fqdn,search.webstie.net
hash:sha256,7fd526e1a190c10c060bac21de17d2c90eb2985633c9ab74020a2b78acd8a4c8
inet:fqdn,dns.domain-resolve.org
hash:sha256,b20327c03703ebad191c0ba025a3f26494ff12c5908749e33e71589ae1e1f6b3
hash:sha256,b214c7a127cb669a523791806353da5c5c04832f123a0a6df118642eee1632a3
hash:sha256,b4e3b2a1f1e343d14af8d812d4a29440940b99aaf145b5699dfe277b5bfb8405
inet:url,http://search.webstie.net/
```

Export Example 2

For this example, we will export the DNS A records we imported in *Ingest Example 1*. We will create a CSV file that matches the format of our original ingest file, with columns for domain, IP, and first / last resolution times.

Stormfile:

To lift the DNS A records for the domains `woot.com`, `hurr.net`, and `derp.org`, we can use the following Storm query:

```
inet:dns:a:fqdn=woot.com inet:dns:a:fqdn=hurr.net inet:dns:a:fqdn=derp.org
```

In this case we want `$lib.csv.emit()` to include:

- the domain (`:fqdn` property of the `inet:dns:a` node).
- the IP (`:ipv4` property of the `inet:dns:a` node).
- the first observed resolution (the first half of the `.seen` property).
- the most recently observed resolution (the second half of the `.seen` property).

As a first attempt, we could specify our output format as follows to export those properties:

```
$lib.csv.emit(:fqdn, :ipv4, .seen)
```

This exports the data from the relevant nodes as expected, but does so in the following format:

```
woot.com,16909060,"(1524057167000, 1529747112000)"
```

We have a few potential issues with our current output:

- The IP address is exported using its raw integer value instead of in human-friendly dotted-decimal format.
- The `.seen` value is exported into a single field as a combined `"(<min>, <max>)"` pair, not as individual comma-separated timestamps.
- The `.seen` values are exported using their raw Epoch millis format instead of in human-friendly datetime strings.

We need to do some additional formatting to get the output we want in the CSV file.

IP Address

Synapse stores IP addresses as integers, so specifying `:ipv4` for our output definition gives us the raw integer value for that property. If we want the human-readable value, we need to use the human-friendly representation (*Repr*) of the value. We can do this using the *$node.repr()* method to tell Storm to obtain and use the repr value of a node instead of its raw value (*$node.value()*).

`$node.repr()` by itself (e.g., with no parameters passed to the method) returns the repr of the primary property value of the node passing through the runtime. Our original Storm query, above, lifts DNS A records - so the nodes passing through the runtime are `inet:dns:a` nodes, not IPv4 nodes. This means that using `$node.repr()` by itself will return the repr of the `inet:dns:a` node, not the `:ipv4` property.

We can tell `$node.repr()` to return the repr of a specific secondary property of the node by passing the **string** of the property name to the method:

```
$node.repr(ipv4)
```

.seen times

`.seen` is an *ival* (interval) type whose property value is a paired set of minimum and maximum timestamps. To export the minimum and maximum as separate fields in our CSV file, we need to split the `.seen` value into two parts by assigning each timestamp to its own variable. We can do this as follows:

```
($first, $last) = .seen
```

However, simply splitting the value will result in the variables `$first` and `$last` storing (and emitting) the raw Epoch millis value of the time, not the human-readable repr value. Similar to the way in which we obtained the repr value for the `:ipv4` property, we need to assign the human-readable repr values of the `.seen` property to `$first` and `$last`:

```
($first, $last) = $node.repr(".seen")
```

Stormfile

We can now combine all of these elements into a Storm query that:

- Lifts the `inet:dns:a` nodes we want to export.
- Splits the human-readable version of the `.seen` property into two time values and assigns them to variables.
- Generates `$lib.csv.emit()` messages to create the CSV rows.

Our full stormfile query looks like this:

```
inet:dns:a:fqdn=woot.com inet:dns:a:fqdn=hurr.net inet:dns:a:fqdn=derp.org

($first, $last) = $node.repr(".seen")

$lib.csv.emit(:fqdn, $node.repr(ipv4), $first, $last)
```

> **Warning:** The data submitted to `$lib.csv.emit()` to create the CSV rows **must** exist for every node processed by the function. For example, if one of the `inet:dns:a` nodes lifted by the Storm query and submitted to `$lib.csv.emit()` does not have a `.seen` property, Storm will generate an error and halt further processing, which may result in a partial export of the desired data.
>
> Subqueries (*Storm Reference - Subqueries*) or various flow control processes (*Storm Reference - Advanced - Control Flow*) can be used to conditionally account for the presence or absence of data for a given node.

Exporting the Data:

We can now test our export of the data we ingested in *Ingest Example 1* (replace the Cortex path below with the path to your Cortex; line is wrapped for readability):

```
python -m synapse.tools.csvtool --debug --export
  --cortex tcp://cortex.vertex.link:4444/cortex00 stormfile export.csv
```

If we view the contents of `export.csv`, we should see the following:

```
woot.com,1.2.3.4,2018/04/18 13:12:47.000,2018/06/23 09:45:12.000
hurr.net,5.6.7.8,2018/10/03 00:47:29.000,2018/10/04 18:26:06.000
derp.org,4.4.4.4,2019/06/09 09:00:18.000,2019/07/03 15:07:52.000
```

```
Detected 159 deprecated properties unlocked and not in use, recommend locking (https://v.
↪vtx.lk/deprlock).
```

```
WARNING: "cmdr" is deprecated in 2.164.0 and will be removed in 3.0.0
```

3.5.6 genpkg

The Synapse `genpkg` tool can be used to generate a Storm *Package* containing new Storm commands and Storm modules from a YAML definition and optionally push it to a Cortex or PkgRepo.

Syntax

`genpkg` is executed from an operating system command shell. The command usage is as follows:

```
usage: synapse.tools.genpkg [-h] [--push <url>] [--save <path>] [--optic <path>]
↪<pkgfile>
```

Where:

- `pkgfile` is the path to the Storm Package YAML file.
- `--save` takes a file name to save the completed package JSON as.
- `--push` takes an optional Telepath URL to a Cortex or PkgRepo for the package to be pushed to.
- `--optic` takes an optional path to a directory containing Optic module files.

Package Layout

The expected filesystem layout for a Storm package is:

```
foopkg.yml
storm/
├── commands/
│   └── foocmd
├── modules/
│   └── foomod
└── optic/
    └── index.html
```

Commands and modules defined in the package YAML file are expected to have corresponding files containing the Storm code for their implementation. It is not required to have both commands and modules in a Storm package; you may have a package with only commands, or only modules.

Package YAML

A Storm package YAML may contain the following definitions:

- `name`: Name of the Storm package.
- `version`: Version of the Storm package. A Cortex may contain multiple versions of the same package.
- `synapse_version`: Optional version specifier for the required Synapse version a Cortex must be running to load the package.
- `onload`: Optional Storm code to run in a Cortex when the package is loaded.
- `modules`: Storm module definitions.
- `commands`: Storm command definitions.

The example below shows the YAML included in the `foopkg.yml` file.

`foopkg.yml`

```
name: foopkg
version: 1.0.0
synapse_version: '>=2.144.0,<3.0.0'

onload: $lib.import(foomod).onload()

modules:
  - name: foomod
    modconf:
      srcguid: f751f9ad20e75547be230ae1a425fb9f

commands:
  - name: foocmd
    descr: |
      One line description on the first line.
      Followed by a more detailed description talking about what the command does and any
      useful additional information.

      Examples:
        # A couple examples of the command
        inet:ipv4 | foocmd
        inet:ipv4 | limit 1 | foocmd --yield
    asroot: true
    cmdargs:
      - - --debug
        - default: false
          action: store_true
          help: Show verbose debug output.

      - - --yield
```

(continues on next page)

(continued from previous page)


```
        - default: false
          action: store_true
          help: Yield the newly created nodes.

      - - --timeout
        - default: 0
          type: int
          help: Specify a timeout in seconds.
    cmdconf:
      srcguid: f751f9ad20e75547be230ae1a425fb9f
    forms:
      input:
        - inet:ipv4
      output:
        - inet:ipv4
      nodedata:
        - [ foodata, file:bytes ]
```


Modules

Modules can be used to expose reusable Storm functions. Each module defines a `name`, which is used for importing elsewhere via `$lib.import()`, and optionally a `modconf` dictionary containing additional configuration values which will be accessible in the module's Storm via `$modconf`.

The example below shows the Storm code included in the `foomod` file.

`foomod`

```
function onload() {
    [ meta:source=$modconf.srcguid
          :name="foomod"
          :type="foo"
    ]
    fini { return($lib.null) }
}

function bar(x, y) {
    return ($($x + $y))
}
```

Commands

Multiple Storm commands can be added to a Storm service package, with each defining the following attributes:

- `name`: Name of the Storm command to expose in the Cortex.
- `descr`: Description of the command which will be available in `help` displays.
- `asroot`: Whether the command should be run with root permissions. This allows users to be granted access to run the command without requiring them to have all the permissions needed by the Storm command. An example `asroot` permission for `foocmd` would be `('storm', 'asroot', 'cmd', 'asroot', 'foocmd')`.
- `cmdargs`: An optional list of arguments for the command.
- `cmdconf`: An optional dictionary of additional configuration variables to provide to the command Storm execution.
- `forms`: List of input and output forms for the command, as well as a list of nodedata keys and the corresponding form on which they may be set by the service.

The example below shows the Storm code included in the `foocmd` file.

foocmd

```
$foo = $lib.import(foomod)

[:asn = $foo.bar(:asn, $(20))]

$node.data.set(foodata, $lib.time.now())
```

Building the Example Package

To build the package and push it directly to a Cortex:

```
python -m synapse.tools.genpkg --push tcp://user:pass@127.0.0.1:27492 foopkg.yml
```

Note: Users must have the `pkg.add` permission to add a package to a Cortex.

Once the package has been successfully pushed to the Cortex, the additional Storm Commands will be listed in the output of `storm help` under the package they were loaded from:

```
package: foopkg
foocmd                      : One line description on the first line.
```

The new commands may now be used like any other Storm command:

```
cli> storm inet:ipv4=192.168.0.113 | foocmd
```

```
Executing query at 2024/04/19 14:25:48.437
```

```
...
```

```
.
```

```
inet:ipv4=192.168.0.113
```

```
.created = 2024/04/19 14:25:48.419
```

```
:asn = 40
```

```
:type = private
```

```
complete. 1 nodes in 51 ms (19/sec).
```

If immediately pushing the package to a Cortex is not desired, it can instead be built and saved to `foo.json` to load later:

```
python -m synapse.tools.genpkg --save foo.json foopkg.yml
```

3.5.7 easycert

The Synapse `easycert` tool can be used to manage CA, host, and user certificates.

Syntax

`easycert` is executed using `python -m synapse.tools.easycert`. The command usage is as follows:

```
usage: easycert [-h] [--certdir CERTDIR] [--importfile {cas,hosts,users}] [--ca] [--p12]␣
↪[--server] [--server-sans SERVER_SANS] [--csr] [--sign-csr] [--signas SIGNAS]
                name

Command line tool to generate simple x509 certs

positional arguments:
  name                  common name for the certificate (or filename for CSR signing)

optional arguments:
  -h, --help            show this help message and exit
  --certdir CERTDIR     Directory for certs/keys
  --importfile {cas,hosts,users}
                        import certs and/or keys into local certdir
  --ca                  mark the certificate as a CA/CRL signer
  --p12                 mark the certificate as a p12 archive
  --server              mark the certificate as a server
  --server-sans SERVER_SANS
                        server cert subject alternate names
  --csr                 generate a cert signing request
  --sign-csr            sign a cert signing request
  --signas SIGNAS       sign the new cert with the given cert name
```

3.6 Storm Reference

Synapse uses the Storm Query language to do lifting and modification of data in the graph. Basic Storm usage is documented in the following sections.

3.6.1 Storm Reference - Introduction

Storm is the query language used to interact with data in Synapse. Storm allows you to ask about, retrieve, annotate, add, modify, and delete data within a Synapse Cortex. If you are using the open source or Quickstart versions of Synapse, you will access Synapse via the Storm command-line interface (**Storm CLI**) (see *storm*):

```
storm> <query>
```

If you are a Synapse Enterprise customer or have requested a Synapse Enterprise demo instance you will access Synapse via the Synapse UI (also known as Optic) and use Storm from the Storm query bar.

Tip: If you're not sure which version of Synapse to start with, check out our Getting Started guide.

Storm Background

In designing Storm, we needed it to be flexible and powerful enough to allow interaction with large amounts of data and a wide range of disparate data types. However, we also needed Storm to be intuitive and efficient so it would be accessible to a wide range of users. We wrote Storm specifically to be used by analysts and other users from a variety of knowledge domains who are not necessarily programmers and who would not want to use what felt like a "programming language".

Wherever possible, we masked Storm's underlying programmatic complexity. The intent is for Storm to act more like a "data language", allowing users to:

- **Reference data and query operations in an intuitive form.** We took a "do what I mean" approach for how users interact with and use Storm so that users can focus on the **data** and the relationships among the data, not the query language. Once you get the gist of it, Storm "just works"! This is because Storm and Synapse make use of a number of features "under the hood" such as property normalization, type enforcement / type awareness, and syntax and query optimization, to make Storm easier for you to use. Synapse and Storm do the work in the background so you can focus on analysis.
- **Use a simple yet powerful syntax to run Storm queries.** Storm uses intuitive keyboard symbols (such as an "arrow" (->) for pivot operations) for efficient querying, as well as a natural language-like syntax. This makes using Storm feel more like "asking a question" than "constructing a data query". In fact, one method we use to teach Storm to new users is to practice "translating" questions into queries (you'll be surprised how straightforward it is!).

Analysts still need to learn the Storm "language" - forms (*Form*) and tags (*Tag*) are Storm's "words", and Storm operators allows you to construct "sentences". That said, the intent is for Storm to function more like "how do I ask this question about the data?" and not "how do I write a program to get the data I need?"

Finally – and most importantly – **giving analysts direct access to Storm allows them to create arbitrary queries and provides them with an extraordinarily powerful analytical tool.** Analysts are not constrained to a set of "canned" queries provided through a GUI or an API. Instead, they can follow their analysis wherever it takes them, creating queries as needed and working with the data in whatever manner is most appropriate to their research.

Basic Storm Operations

Storm allows users to perform all of the common operations used to interact with data in Synapse:

- Lift: retrieve data based on specified criteria.
- Filter: refine your results by including or excluding a subset of nodes based on specified criteria.
- Pivot: take a set of nodes and identify other nodes that share one or more property values with the lifted set.
- Traverse light edges.
- Modify data: create, modify, annotate (tag), and delete nodes from Synapse.
- Run commands: Storm supports an extensible set of commands. Many commands provide specific functionality to extend the analytical power of Storm. Other Storm commands allow management of permissions for users and roles, Synapse views and layers, and Synapse's automation features. You can display available commands by running `help` from the Storm CLI.

Many Storm queries - even "complex" ones - can be constructed from this simple set of "building blocks". For users who want to expand their Storm capabilities, there are additional *Advanced Storm Operations* that provide even greater power and flexibility.

Lift, Filter, and Pivot Criteria

The main operations carried out with Storm are lifting, filtering, and pivoting (we include traversing light edges as part of "pivoting"). When conducting these operations, you need to be able to clearly specify the data you are interested in – your selection criteria. In most cases, the criteria you specify will be based on one or more of the following:

- A **property** (primary or secondary) on a node.
- A specific **value** for a property (*<form>* = *<valu>* or *<prop>* = *<pval>*) on a node.
- A **tag** on a node.
- The existence of a **light edge** linking nodes.
- The name ("verb") of a specific **light edge** linking nodes.

All of the above elements – nodes, properties, values, and tags – are the fundamental building blocks of the Synapse data model (see *Data Model Objects*). **As such, an understanding of the Synapse data model is essential to effective use of Storm.**

Whitespace and Literals in Storm

The Storm query language allows (and in some cases requires) whitespace in order to separate syntax elements such as commands and command arguments.

When using **literals** in Storm, quotation marks are used to **preserve** whitespace characters and other special characters within the literal.

Using Whitespace Characters

Whitespace characters (i.e., spaces) are used within Storm to separate command line arguments. Specifically, whitespace characters are used to separate commands, command arguments, command operators, variables and literals.

When entering a query/command in Storm, one or more whitespace characters are **required** between the following command line arguments:

- A command (such as `max`) and command line parameters (in this case, the property `:asof`):

```
storm> inet:whois:rec:fqdn=vertex.link | max :asof
```

- An unquoted literal and any subsequent argument or operator:

```
storm> inet:email=support@vertex.link | count

storm> inet:email=support@vertex.link -> *
```

Whitespace characters can **optionally** be used when performing the following operations:

- Assigning values using the equals sign assignment operator:

```
storm> [inet:ipv4=192.168.0.1]

storm> [inet:ipv4 = 192.168.0.1]
```

- Comparison operations:

```
storm> file:bytes:size>65536

storm> file:bytes:size > 65536
```

- Pivot operations:

```
storm> inet:ipv4->*

storm> inet:ipv4 -> *
```

- Specifying the content of edit brackets or edit parentheses:

```
storm> [inet:fqdn=vertex.link]

storm> [ inet:fqdn=vertex.link ]

storm> [ inet:fqdn=vertx.link (inet:ipv4=1.2.3.4 :asn=5678) ]

storm> [ inet:fqdn=vertex.link ( inet:ipv4=1.2.3.4 :asn=5678 ) ]
```

Whitespace characters **cannot** be used between reserved characters when performing the following CLI operations:

- Add and remove tag operations. The plus (+) and minus (-) sign characters are used to add and remove tags to and from nodes in Synapse respectively. When performing tag operations using these characters, a whitespace character cannot be used between the actual character and the tag name (e.g., `+#<tag>`).

```
storm> inet:ipv4 = 192.168.0.1 [ -#oldtag +#newtag ]
```

Entering Literals

Storm uses quotation marks (single and double) to preserve whitespace and other special characters that represent literals. If values with these characters are not quoted, Synapse may misinterpret them and throw a syntax error.

Single (`' '`) or double (`" "`) quotation marks can be used when specifying a literal in Storm during an assignment or comparison operation. Enclosing a literal in quotation marks is **required** when the literal:

- begins with a non-alphanumeric character,
- contains a space (`\s`), tab (`\t`) or newline(`\n`) character, or
- contains a reserved Synapse character (for example, `\ ) , = ] } |`).

Enclosing a literal in **single** quotation marks will preserve the literal meaning of **each character.** That is, each character in the literal is interpreted exactly as entered.

  - Note that if a literal (such as a string) **includes** a single quotation mark / tick mark, it must be enclosed in double quotes.
  - Wrong: `'Storm's intuitive syntax makes it easy to learn and use.'`
  - Right: `"Storm's intuitive syntax makes it easy to learn and use."`

Enclosing a literal in **double** quotation marks will preserve the literal meaning of all characters **except for** the backslash (`\`) character, which is interpreted as an 'escape' character. The backslash can be used to include special characters such as tab (`\t`) or newline (`\n`) within a literal.

  - If you need to include a literal backslash within a double-quoted literal, you must enter it as a "double backslash" (the first backslash "escapes" the following backslash character):
    - Wrong: `"C:\Program Files\Mozilla Firefox\firefox.exe"`
    - Right: `"C:\\Program Files\\Mozilla Firefox\\firefox.exe"`

  Note that because the above example does not include a single quote / tick mark as part of the literal, you can simply enclose the file path in single quotes:

  - Also right: `'C:\Program Files\Mozilla Firefox\firefox.exe'`

The Storm queries below demonstrate assignment and comparison operations that **do not require** quotation marks:

- Lifting the domain `vtx.lk`:

```
storm> inet:fqdn = vtx.lk
```

- Lifting the file name `windowsupdate.exe`:

```
storm> file:base = windowsupdate.exe
```

The commands below demonstrate assignment and comparison operations that **require** the use of quotation marks. Failing to enclose the literals below in quotation marks will result in a syntax error.

- Lift the file name `windows update.exe` which contains a whitespace character:

```
storm> file:base = 'windows update.exe'
```

- Lift the organization name `The Vertex Project, LLC` which contains both whitespace and the comma special character:

```
storm> ou:name = 'The Vertex Project, LLC'
```

Backtick Format Strings

Backticks (` `) can be used to specify a format string in Storm, with curly braces used to specify expressions which will be substituted into the string at runtime. Any valid Storm expression may be used in a format string, such as variables, node properties, tags, or function calls.

- Use a variable in a string:

```
storm> $ip = "1.2.3.4" $str = `The IP is {$ip}`
```

- Use node properties in a string:

```
storm> inet:ipv4=1.2.3.4 $lib.print(`IP {$node.repr()}: asn={:asn} .seen={.seen} foo={
↪#foo}`)
```

- Lift a node using a format string:

```
storm> $ip=1.2.3.4 $port=22 inet:client=`{$ip}:{$port}`
```

Backtick format strings may also span multiple lines, which will include the newlines when displayed:

```
storm> inet:ipv4=1.2.3.4 $lib.print(`
IP {$node.repr()}:
asn={:asn}
.seen={.seen}
foo={#foo}`)
```

Like double quotes, backticks will preserve the literal meaning of all characters **except for** the backslash (\ ) character, which is interpreted as an 'escape' character. The backslash can be used to include special characters such as tab (`\t`) or newline (`\n`), or to include a backtick (`) or curly brace ({) in the string.

Storm Operating Concepts

Storm has several notable features in the way it interacts with and operates on data. These concepts are important but also pretty intuitive; it's good to be familiar with them, but most users don't need to worry about them too much for standard Storm queries and operations (day-to-day interaction with Synapse data).

These concepts are much more important if you're using more advanced Storm constructs such as variables, control flow, or functions. If you're writing advanced Storm queries, automation, or custom Power-Ups, you should be comfortable with these terms and behaviors.

Working Set

Most objects in Synapse are **nodes**. Most Storm operations start by **lifting** (selecting) a node or set of nodes from Synapse's data store.

- The set of nodes that you start with is called your **initial working set**.
- The set of nodes at any given point in your Storm query is called your **current working set**.

Operation Chaining

Users commonly interact with data (nodes) in Synapse using operations such as lift, filter, and pivot. Storm allows multiple operations to be **chained** together to form increasingly complex queries:

```
storm> inet:fqdn=vertex.link

storm> inet:fqdn=vertex.link -> inet:dns:a

storm> inet:fqdn=vertex.link -> inet:dns:a -> inet:ipv4

storm> inet:fqdn=vertex.link -> inet:dns:a -> inet:ipv4 +:type=unicast
```

The above example demonstrates chaining a lift (`inet:fqdn=vetex.link`) with two pivots (`-> inet:dns:a`, `-> inet:ipv4`) and a filter (`+:type=unicast`).

When Storm operations are concatenated in this manner, they are processed **in order from left to right** with each operation (lift, filter, or pivot) acting on the output of the previous operation. A Storm query is not evaluated as a single whole; Storm evaluates your working set of nodes against each operation in order before moving to the next operation.

Note: Technically, any query you construct is first evaluated as a whole **to ensure it is a syntactically valid query** - Synapse will complain if your Storm syntax is incorrect. But once Synapse has checked your Storm syntax, nodes are processed by each Storm operation in order.

You do not have to write (or execute) Storm queries "one operation at a time" - this example is meant to illustrate how you can chain individual Storm operations together to form longer queries. If you know that the question you want Storm to answer is "show me the unicast IPv4 addresses that the FQDN vertex.link has resolved to", you can simply run the final query in its entirety. But you can also "build" queries one operation at a time if you're exploring the data or aren't sure yet where your analysis will take you.

The ability to build queries operation by operation means that a Storm query can parallel an analyst's natural thought process: you perform one Storm operation and then consider the "next step" you want to take in your analysis!

Node Consumption

Storm operations typically **transform** your working set in some way. That is, the nodes that "go into" (are inbound) to a given Storm operation are not necessarily the nodes that "come out" of that operation.

Take our operation chaining example above:

- Our **initial working set** consists of the single node `inet:fqdn=vertex.link`, which we selected with a lift operation.
- When we pivot to the DNS A records for that FQDN, we navigate away from (drop) our initial `inet:fqdn` node, and navigate to (add) the DNS A nodes. Our **current working set** now consists of the DNS A records (`inet:dns:a` nodes) for vertex.link.
- Similarly, when we pivot to the IPv4 addresses, we navigate away from (drop) the DNS A nodes and navigate to (add) the IPv4 nodes. Our current working set is made up of the `inet:ipv4` nodes.
- Finally, when we perform our filter operation, we may discard (drop) any IPv4 nodes representing non-unicast IPs (such as `inet:ipv4=127.0.0.1`) if present.

We refer to this transformation (in particular, dropping) of some or all nodes by a given Storm operation as **consuming** nodes. Most Storm operations consume nodes (that is, change your working set in some way - what comes out of the operation is not the same set of nodes that goes in).

For standard Storm queries this process should be fairly intuitive ("now that you point that out. . . of course that is what's happening"). However, the idea of **node consumption** and the transformation of your current working set is important to keep in mind for more advanced Storm.

Tip: Storm commands (built-in commands, or commands added by Power-Ups) that operate on nodes generally do **not** consume nodes - the nodes that "go into" the command are the same nodes that "come out" by default. This allows you to chain multiple commands together that all operate on the same inbound nodes. Commands may include a `--yield` option to modify this behavior and drop (consume) the inbound nodes and return the node(s) (or primary node(s)) produced by the command.

Storm as a Pipeline

Just as each Storm **operation** in the chain is processed individually from left to right, **each node** in your working set is evaluated **individually** against a given Storm operation in a query. You can think of your Storm query as a **pipeline** of operations, with each node "fired" one at a time through the pipeline. Whether you start with one node or 10,000 nodes, they are evaluated against your Storm query one by one.

Processing nodes one by one significantly reduces Synapse's latency and memory use: this is a big part of what makes Synapse so fast and responsive. Synapse can immediately provide you with results for the initial nodes while it continues processing the remaining nodes. In other words, you don't have to wait for your **entire** query to complete before starting to see results.

For everyday Storm, this behavior is transparent - you run a Storm query, you get a response. However, understanding this pipeline behavior is critical when working with (or troubleshooting) Storm queries that leverage features such as subqueries, variables, control flow operations, or functions.

Advanced Storm Operations

In our experience, the more analysts use Storm, the more they want even greater power and flexibility from the language to support their analytical workflow! To meet these demands, Storm evolved a number of advanced features, including:

- Variables
- Methods
- Control Flow
- Functions
- *Storm Libraries*
- *Storm Types*

Analysts do not need to use or understand these more advanced concepts in order to use Storm or Synapse. Basic Storm functions are sufficient for a wide range of analytical needs and workflows. However, these additional features are available to both "power users" and developers as needed:

- For analysts, once they are comfortable with Storm basics, many of them want to expand their Storm skills **specifically because it facilitates their analysis.**
- For developers, writing extensions to Synapse in Storm has the advantage that the extension **can be deployed or updated on the fly.** Contrast this with extensions written in Python, for example, which would require restarting the system during a maintenance window in order to deploy or update the code.

Note: Synapse's Rapid Power-Ups are written entirely in Storm and exposed to Synapse users as Storm commands!

3.6.2 Storm Reference - Document Syntax Conventions

This section covers the following important conventions used within the Storm Reference Documents:

- *Storm and Layers*
- *Storm Syntax Conventions*
- *Usage Statements vs. Specific Storm Queries*
- *Type-Specific Behavior*
- *Whitespace*

Storm and Layers

The Storm Reference documentation provides basic syntax examples that assume a simple Storm environment - that is, a Cortex with a single Layer. For multi-Layer Cortexes, the effects of specific Storm commands - particularly data modification commands - may vary based on the specific arrangement of read / write Layers, the Layer in which the command is executed, and the permissions of the user.

Storm Syntax Conventions

The Storm Reference documentation provides numerous examples of both abstract Storm syntax (usage statements) and specific Storm queries. The following conventions are used for Storm **usage statements:**

- Items that must be entered literally on the command line are in **bold.** These items include command names and literal characters.
- Items that represent "variables" that must be replaced with a name or value are placed within angle brackets (< >) in *italics*. Most "variables" are self-explanatory, however a few commonly used variable terms are defined here for convenience:
  - <*form*> refers to a form / node primary property, such as `inet:fqdn`.
  - <*valu*> refers to the value of a primary property, such as `woot.com` in `inet:fqdn=woot.com`.
  - <*prop*> refers to a node secondary property (including universal properties) such as `inet:ipv4:asn` or `inet:ipv4.created`.
  - <*pval*> refers to the value of a secondary property, such as `4808` in `inet:ipv4:asn=4808`.
  - <*query*> refers to a Storm query.
  - <*inet:fqdn*> refers to a Storm query whose results contain the specified form(s)
  - <*tag*> refers to a tag (`#sometag` as opposed to a `syn:tag` form).
- **Bold brackets** are literal characters. Parameters enclosed in non-bolded brackets are optional.
- Parameters **not** enclosed in brackets are required.
- A vertical bar signifies that you choose only one parameter. For example:
  - `a | b` indicates that you must choose a or b.
  - `[ a | b ]` indicates that you can choose a, b, or nothing (the non-bolded brackets indicate the parameter is optional).
- Ellipses (...) signify the parameter can be repeated on the command line.
- The `storm` command that must precede a Storm query is assumed and is omitted from examples.

Example:

[*<form>* **=** *<valu>* [**:** *<prop>* **=** *<pval>* ...] **]**

The Storm query above adds a new node.

- The outer brackets are in **bold** and are required literal characters to specify a data modification (add) operation. Similarly, the equals signs are in **bold** to indicate literal characters.
- *<form>* and *<valu>* would need to be replaced by the specific form (such as `inet:ipv4`) and primary property value (such as `1.2.3.4`) for the node being created.
- The inner brackets are not bolded and indicate that one or more secondary properties can **optionally** be specified.
- *<prop>* and *<pval>* would need to be replaced by the specific secondary property and value to add to the node, such as `:loc = us`.
- The ellipsis (`...`) indicate that additional secondary properties can optionally be specified.

Usage Statements vs. Specific Storm Queries

Examples of specific queries represent fully literal input, but are not shown in bold for readability. For example:

Usage statement:

[*<form>* **=** *<valu>* [**:** *<prop>* **=** *<pval>* ...] **]**

Example query:

[inet:ipv4 = 1.2.3.4 :loc = us]

Type-Specific Behavior

Some data types within the Synapse data model have been optimized in ways that impact their behavior within Storm queries (e.g., how types can be input, lifted, filtered, etc.) See *Storm Reference - Type-Specific Storm Behavior* for details.

Whitespace

Whitespace may be used in the examples for formatting and readability.

3.6.3 Storm Reference - Lifting

Lift operations retrieve a set of nodes based on the specified criteria. While all lift operations are retrieval operations, they can be broken down into "types" of lifts based on the criteria, comparison operator, or special handler used:

- *Lift by Form*
- *Lift by Property*
- *Lift by Property Value - Standard Comparison Operators*
- *Lift by Property Value - Extended Comparison Operators*
- *Tag Lifts*

In addition, the specialized "reverse" keyword and the "try" operator can each be used with lift operations to modify their behavior:

- *"reverse" Keyword*

- *"Try" Operator*

Tip: When performing lift operations, you can specify the name of an *Interface* to represent all forms that inherit that interface. See the sections below for details and examples.

See *Storm Reference - Document Syntax Conventions* for an explanation of the syntax format used below.

See *Storm Reference - Type-Specific Storm Behavior* for details on special syntax or handling for specific data types.

Lift by Form

"Lift by form" operations return **all** nodes of that form in Synapse. The wildcard (asterisk) character (`*`) can be used to lift all nodes of all forms that match a partial form name / namespace.

If a form inherits an *Interface*, you can specify the interface name to lift all nodes of all forms that inherit the interface.

Tip: In a production instance of Synapse, lifting **all** nodes of a commonly used form (e.g., `inet:fqdn` or `inet:ipv4`) or lifting by an interface that is inherited by numerous forms (e.g. `it:host:activity`) may return thousands or tens of thousands of nodes. Lifting by form or interface can be used with the Storm *limit* command to return only a specified number of nodes (for example, to view a sample of available data).

Lift by Form Name

A "lift by form name" operation returns all nodes for the specified *Form*. This type of lift requires the name of the form whose nodes you want to lift.

Syntax:

<form>

Examples:

Lift all FQDN (inet:fqdn) nodes:

```
inet:fqdn
```

Lift all nodes representing articles (media:news nodes):

```
media:news
```

Lift by Form Name - Wildcard

You can use the wildcard (asterisk) character (`*`) to specify all forms that match a partial form name. Use of the wildcard is not limited to form namespace boundaries.

Note: The wildcard can only be used at the end of the partial match. It cannot be used at the beginning or in the middle of the form name. For example, both of the following are **invalid**:

`*:header`

`it:exec:*:del`

In addition, use of the wildcard does not extetnd to partial matching of property names. For example, the following is **invalid**:

`file:bytes:mime:pe:*`

Syntax:

<partial_form_name> *****

Examples:

Lift all nodes in the MITRE ATT&CK form namespace (e.g., it:mitre:attack:group, it:mitre:attack:technique, etc.):

```
it:mitre:attack:*
```

Lift all DNS A (inet:dns:a) and DNS AAAA (inet:dns:aaaa) nodes:

```
inet:dns:a*
```

Lift Form by Interface

You can use the name of an *Interface* to lift all forms that inherit that interface.

Note: When lifting by interface, you cannot use the wildcard (*) character to match multiple interface names. Synapse will interpret use of the wildcard as an attempt to match multiple form names.

Syntax:

<interface>

Examples:

Lift all host activity nodes (all nodes of all forms that inherit it:host:activity interface):

```
it:host:activity
```

Lift all taxonomy nodes (all nodes of all forms that inherit the meta:taxonomy interface):

```
meta:taxonomy
```

Lift by Property

A "lift by property" operation returns all nodes that **have** the specified *Property* set, regardless of the property value. In most cases, this type of lift requires the full name (i.e., the combined form and property name) of the property you want to use to lift the nodes. When lifting by universal property, the form name is only needed if you want to lift nodes of a specific form that have the universal property.

Lift by Secondary Property

Syntax:

<form> **:** *<prop>*

Examples:

Lift all IPv4 (inet:ipv4) nodes that have a location (:loc property):

```
inet:ipv4:loc
```

Lift all file (file:bytes) nodes that have a PDB path (:mime:pe:pdbpath property):

```
file:bytes:mime:pe:pdbpath
```

Lift by Interface Property

If a form inherits an *Interface*, you can lift all nodes of all forms that have an interface-derived property by specifying the full name of the interface and its property.

Syntax:

<interface> **:** *<prop>*

Examples:

Lift all host activity nodes (all nodes of all forms that inherit the it:host:activity interface) that have a :time property:

```
it:host:activity:time
```

Lift all protocol request nodes (all nodes of all forms that inherit the inet:proto:request interface) that have an associated network flow (i.e., that have a :flow property):

```
inet:proto:request:flow
```

Lift by Universal Property

Syntax:

[*<form>*] **.** *<prop>*

A *Universal Property* applies to and can be used by any node. Synapse uses two universal properties:

- `.created`, a time (date/time) value representing when a node was created in Synapse.
- `.seen`, an interval (pair of date/time values) that can be used optionally to represent "when" the object represented by a node existed or was observed.

Examples:

Lift all email (inet:email) nodes with a .seen property:

```
inet:email.seen
```

Lift all nodes with a .seen property:

```
.seen
```

Lift all nodes in Synapse:

```
.created
```

Tip: Because the `.created` property is automatically set for every node when it is first added to Synapse, lifting by the `.created` property effectively lifts every node in Synapse (technically, every node in the current *View*).

Lift by Extended Property

Syntax:

<form> **:_** *<prop>*

An *Extended Property* is a custom property that has been added to the Synapse data model to capture specialized data for a given form. To avoid potential namespace collisions with standard properties, extended property names must begin with an underscore. In addition, we recommend using the name of the source or vendor of the property data as the first property namespace element.

Examples:

Lift all of the files (file:bytes) nodes that have a VirusTotal "reputation" extended property (:_virustotal:reputation property):

```
file:bytes:_virustotal:reputation
```

Note: The `:_virustotal:reputation` extended property is added to the Synapse data model by the Synapse-VirusTotal Power-Up.

Lift by Property Value - Standard Comparison Operators

A "lift by property value" operation returns the node(s) whose property matches the specified value. This type of lift requires:

- the form name or full property name (i.e., the combined form and property name) that you will use to lift the node(s);
- a *Comparison Operator* to specify how the property value should be evaluated; and
- the property value.

A "lift by property value" can be performed using primary, secondary, universal, or extended properties.

Tip: When lifting by a secondary property value, you can specify either a form name or an *Interface* name.

In Synapse, we define **standard comparison operators** as the following set of operators:

- equal to (=)
- less than (<)
- greater than (>)

- less than or equal to (<=)
- greater than or equal to (>=)

The *"Try" Operator* (?=) can optionally be used in place of the standard equal to operator (=). Use of the try operator is generally not required for interactive Storm queries, but may be useful for more complex Storm queries (such as automation or Storm-based ingest queries).

The most commonly used standard comparison operator is the equal to (=) operator. Comparison operators that expect a **quantity** (i.e., the inequality symbools <, >, <=, and >=) can only be used with properties whose type supports the comparison (e.g., integers, dates/times, etc.)

Tip: IPv4 addresses (`inet:ipv4` nodes) are stored as their decimal integer equivalents (even though they are displayed in familiar "dotted decimal" format), and can be used with the various inequality operators:

```
inet:ipv4 < 192.168.0.0
```

IPv6 addresses (`inet:ipv6` nodes) are stored as strings due to limitations with msgpack, but can be used with the various inequality operators if enclosed in single or double quotes:

```
inet:ipv6 >= '::0'
```

Lift by Primary Property Value

Syntax:

<form> <operator> <valu>

Examples:

Lift the FQDN vertex.link:

```
inet:fqdn = vertex.link
```

Lift the DNS A record showing that domain woot.com resolved to IP 1.2.3.4:

```
inet:dns:a = (woot.com, 1.2.3.4)
```

Lift the organization node whose primary property is the specified guid (globally unique identifier):

```
ou:org = 4b0c2c5671874922ce001d69215d032f
```

Lift the Autonomous System (inet:asn) nodes whose AS number is less than 50:

```
inet:asn < 50
```

Lift by Secondary Property Value

Syntax:

<form> **:** *<prop> <operator> <pval>*

Examples:

Lift the organization node with the alias "vertex":

```
ou:org:alias = vertex
```

Lift all DNS A records for the FQDN "hugesoft.org":

```
inet:dns:a:fqdn = hugesoft.org
```

Lift all the files with a PE compiled time of 1992-06-19 22:22:17:

```
file:bytes:mime:pe:compiled = '1992/06/19 22:22:17'
```

Lift all the files with a PE compiled time that falls within the year 2019:

```
file:bytes:mime:pe:compiled = 2019*
```

Lift files whose size is less than 1MB:

```
file:bytes:size < 1000000
```

Lift domain WHOIS records for FQDNs registered (created) after January 1, 2023:

```
inet:whois:rec:created > 2023/01/01
```

Lift PE files that were compiled during the year 2012 or earlier:

```
file:bytes:mime:pe:compiled <= 2012*
```

Lift contact data (ps:contact nodes) for individuals where the date of birth (:dob property) is January 1, 1990 or later:

```
ps:contact:dob >= 1990/01/01
```

Usage Notes:

- When lifting nodes by secondary property value where the value is a time (date / time), you do not need to use full `YYYY/MM/DD hh:mm:ss.mmm` syntax. Synapse allows the use of both lower resolution values (e.g., `YYYY/MM/DD`) and wildcard values (e.g., `YYYY/MM*`). In particular, wildcard syntax can be used to specify any values that match the wildcard expression. See the type-specific documentation for *time* types for a detailed discussion of these behaviors.

Lift by Interface Property Value

If a form inherits an *Interface*, you can lift all nodes of all forms with a specific interface-derived property value by using the name of the interface.

Tip: Synapse returns results in lexical order (sorted, ascending to descending) based on the way the queried property is indexed. When using an interface to lift by secondary property, Synapse performs the lifts for each form in parallel, and yields the results in order. See the *"reverse" Keyword* section for additional discussion of this concept.

Syntax:

<interface> **:** *<prop> <operator> <pval>*

Examples:

Lift all host activity nodes (all nodes of all forms that inherit the it:host:activity interface) associated with the host name "ron-pc":

```
it:host:activity:host = { it:host:name=ron-pc }
```

Tip: The `:host` property is a guid value (the guid of an `it:host` node). The example above uses a *Subquery* to specify the host guid value using another Storm query (i.e., "The guid of the it:host node whose :name is ron-pc") insetad of specifying the guid value directly. Subqueries are a useful way to work with guid forms by referencing nodes using more human-friendly secondary property values. See *Using Subqueries to Reference Nodes* for a more detailed discussion.

Lift all host activity nodes (all nodes of all forms that inherit the it:host:activity interface) that were observed on or after February 1, 2024:

```
it:host:activity:time >= 2024/02/01
```

Lift by Universal Property Value

Synapse has two built-in universal properties:

- `.created` (a time) which represents the date and time a node was created in Synapse; and
- `.seen` (an interval), a pair of date / time values that can optionally be used to represent when a node existed or was observed.

Times (date / time values) are stored as integers (epoch milliseconds) in Synapse and can be lifted using any standard comparison operator.

Because intervals are a pair of date / time values, they can only be lifted using the equal to (=) standard comparison operator to specify an **exact** match for the interval values.

The *Lift by Time or Interval (@=)* and *Lift by Range (*range=)* extended comparison operators provide additional flexibility when lifting by times and intervals.

See also the *time* and *ival* sections of the *Storm Reference - Type-Specific Storm Behavior* guide for additional details on working with times and intervals in Synapse.

Syntax:

[*<form>*] **.** *<prop> <operator> <pval>*

Examples:

Lift all nodes created after January 1, 2024:

```
.created >= 2024/01/01
```

Lift all DNS A records (inet:dns:a nodes) whose .seen property exactly matches the specified interval:

```
inet:dns:a.seen = ('2023/11/19 12:11:42.041', '2023/12/27 08:05:47.776')
```

Lift by Extended Property Value

When lifting by extended property value, you can use any standard comparison operator supported by the property's type. For example, if the extended property is a string, only the equal to (=) standard operator is supported. If the extended property is an integer, any of the standard operators can be used.

Syntax:

<form> **:_** *<prop> <operator> <pval>*

Example:

Lift all files with a VirusTotal "reputation" score less than -50:

```
file:bytes:_virustotal:reputation < -50
```

Lift by Property Value - Extended Comparison Operators

Storm supports a set of extended comparison operators (comparators) for specialized lift operations.

- *Lift by Regular Expression (~=)*
- *Lift by Prefix (^=)*
- *Lift by Time or Interval (@=)*
- *Lift by Range (*range=)*
- *Lift by Set Membership (*in=)*
- *Lift by Proximity (*near=)*
- *Lift by (Arrays) (*[ ])*

Just as with standard comparison operators, lifting by property value with extended comparison operators requires:

- the form name or full property name (i.e., the combined form and property name) that you will use to lift the node(s);
- a *Comparison Operator* to specify how the property value should be evaluated; and
- the property value.

Each extended comparison operator can be used with any kind of property (primary, secondary, universal, or extended) whose *Type* is appropriate for the comparison used.

Tip: When lifting by a secondary property value, you can specify either a form name or an *Interface* name.

Synapse returns results in lexical order (sorted, ascending to descending) based on the way the queried property is indexed. When using an interface to lift by secondary property, Synapse performs the lifts for each form in parallel, and yields the results in order. See the *"reverse" Keyword* section for additional discussion of this concept.

Lift by Regular Expression (~=)

The extended comparator ~= is used to lift nodes based on PCRE-compatible regular expressions.

Tip: *Lift by Prefix (^=)* can be used to match the beginning of string-based properties. It is much faster than lifting by regex for start-of-string matches and should be used instead of a regular expression lift where possible.

Syntax:

<form> [**:** | **.** | **:_** *<prop>*] **~=** *<regex>*

<interface> **:** *<prop>* **~=** *<regex>*

Examples:

Lift all files with PDB paths containing the string "rouji":

```
file:bytes:mime:pe:pdbpath ~= rouji
```

Lift all organizations whose name contains a string that starts with "v", followed by 0 or more characters, followed by "x":

```
ou:org:name ~= '^v.*x'
```

Lift by Prefix (^=)

Synapse performs prefix indexing on string and string-derived types, which optimizes lifting nodes whose *<valu>* or *<pval>* starts with a given prefix (substring). The extended comparator ^= is used to lift nodes by prefix.

Note: Extended string types that support dotted notation (such as the *loc* or *syn:tag* types) have custom behaviors with respect to lifting and filtering by prefix.

inet:fqdn nodes are indexed in reverse string order so cannot be lifted using the prefix extended operator. However, reverse indexing allows wildcard (*) matching of the beginning of any FQDN string.

See the relevant sections in the *Storm Reference - Type-Specific Storm Behavior* guide for details.

Syntax:

<form> [**:** | **.** | **:_** *<prop>*] **^=** *<prefix>*

<interface> **:** *<prop>* **^=** *<prefix>*

Examples:

Lift all email addresses that start with "abuse":

```
inet:email ^= abuse
```

Lift all organizations whose name starts with "ministry":

```
ou:org:name ^= ministry
```

Lift all Microsoft Office metadata nodes (all nodes of all forms that inherit the file:mime:msoffice interface) whose :author name starts with "DESKTOP":

```
file:mime:msoffice:author ^= DESKTOP
```

Lift all tags (syn:tag) nodes in the "rep.alienvault" tree where the third tag element starts with the numeral "0":

```
syn:tag ^= rep.alienvault.0
```

Tip: Even though tag elements are dot-separated, when lifting `syn:tag` nodes by prefix the prefix string is not constrained to dot boundaries. In other words, a prefix lift used with tags can match a partial tag element. The query above will match all of the following tags:

- syn:tag=rep.alienvault.0_day
- syn:tag=rep.alienvault.0_days
- syn:tag=rep.alienvault.0day
- syn:tag=rep.alienvault.0days
- syn:tag=rep.alienvault.0ktapus

Lift by Time or Interval (@=)

Many forms include properties that are date / time values (<*ptype*> = <*time*>) or time windows / intervals (<*ptype*> = <*ival*>). The time/interval extended comparator `@=` is used to lift nodes based on comparisons among various combinations of times and intervals.

Tip: See *Storm Reference - Type-Specific Storm Behavior* for additional detail on the use and behavior of *time* and *ival* data types.

Syntax:

<form> **:** | **.** | **:_** *<prop>* **@=(** *<ival_min>* **,** *<ival_max>* **)**

<form> **:** | **.** | **:_** *<prop>* **@=** *<time>*

<interface> **:** *<prop>* **@=(** *<ival_min>* **,** *<ival_max>* **)**

<interface> **:** *<prop>* **@=** *<time>*

Examples:

Lift all DNS A records whose .seen values fall between July 1, 2022 and August 1, 2022:

```
inet:dns:a.seen @= ( 2022/07/01, 2022/08/01 )
```

Lift all DNS requests that occurred on May 3, 2023 (between 05/03/2023 00:00:00 and 05/03/2023 23:59:59):

```
inet:dns:request:time @= ( '2023/05/03 00:00:00', '2023/05/04 00:00:00' )
```

Lift the WHOIS email nodes that were observed between July 1, 2023 and the present:

```
inet:whois:email.seen @= ( 2023/07/01, now )
```

Lift the network flows that occurred within the past day:

```
inet:flow:time @= ( now, '-1 day' )
```

Lift all host activity nodes (all nodes of all forms that inherit the it:host:activity interface) that occurred within the past three hours:

```
it:host:activity:time @= (now, '-3 hours')
```

Usage Notes:

- When specifying an interval with the `@=` operator, the minimum value is included in the interval for comparison purposes but the maximum value is **not**. This is equivalent to "greater than or equal to *<min>* and less than *<max>*". This behavior differs from that of the `*range=` operator, which includes **both** the minimum and maximum.
- **Comparing intervals to intervals:** when using an interval with the `@=` operator to lift nodes based on an interval property, Synapse returns nodes whose interval value has **any** overlap with the specified interval. For example:
  - A lift interval of September 1, 2018 to October 1, 2018 (2018/09/01, 2018/10/01) will match nodes with any of the following intervals:
    - August 12, 2018 to September 6, 2018 (2018/08/12, 2018/09/06).
    - September 13, 2018 to September 17, 2018 (2018/09/13, 2018/09/17).
    - September 30, 2018 to November 5, 2018 (2018/09/30, 2018/11/05).
- **Comparing intervals to times:** When using an interval with the `@=` operator to lift nodes based on a time property, Synapse returns nodes whose time value falls within the specified interval.
- **Comparing times to times:** When using a time with the `@=` operator to lift nodes based on a time property, Synapse returns nodes whose timestamp is an **exact match** of the specified time. In other words, in this case the interval comparator (`@=`) behaves like the equal to comparator (=).
- When specifying date / time and interval values, Synapse allows the use of both lower resolution values (e.g., `YYYY/MM/DD`), and wildcard values (e.g., `YYYY/MM*`). Wildcard time syntax may provide a simpler and more intuitive means to specify some intervals. For example `inet:whois:rec:asof=2018*` is equivalent to `inet:whois:rec:asof@=('2018/01/01', '2019/01/01')`.
- Time-based keywords (such as `now`) and relative time syntax (expressions such as `+-1 hour` or `-7 days`) can be used for interval values.

  See the type-specific documentation for *time* and *ival* types for a detailed discussion of these behaviors.

Lift by Range (*range=)

The range extended comparator (`*range=`) supports lifting nodes whose *<form>* = *<valu>* or *<prop>* = *<pval>* fall within a specified range of values. The comparator can be used with types such as integers and times.

Note: The `*range=` operator can be used to lift both `inet:ipv4` and `inet:ipv6` values (which are stored as decimal integers and strings, respectively). However, ranges of *inet:ipv4* and `inet:ipv6` nodes can also be lifted directly by specifying the lower and upper addresses in the range using `<min>-<max>` format. For example:

`inet:ipv4 = 192.168.0.0-192.168.0.10`

Because IPv6 nodes are stored as strings, the range must be enclosed in quotes:

```
inet:ipv6 = "::0-ff::ff"
```

The `*range=` operator cannot be used to compare a time range with a property value that is an interval (`ival` type). The interval (`@=`) operator should be used instead.

Syntax:

<form> [**:** | **.** | **:_** *<prop>*] ***range = (** *<range_min>* **,** *<range_max>* **)**

<interface> **:** *<prop>* ***range = (** *<range_min>* **,** *<range_max>* **)**

Examples:

Lift all files whose size is between 1000 and 100000 bytes:

```
file:bytes:size *range= ( 1000, 100000 )
```

Lift all files whose VirusTotal "reputation" score is between -20 and 20:

```
file:bytes:_virustotal:reputation *range= ( -20, 20 )
```

Lift all domain WHOIS records that were captured / retrieved between November 29, 2013 and June 14, 2016:

```
inet:whois:rec:asof *range= ( 2013/11/29, 2016/06/14 )
```

Lift all DNS requests made within one day of December 1, 2021:

```
inet:dns:request:time *range= ( 2021/12/01, '+-1 day' )
```

Usage Notes:

- When specifying a range, **both** the minimum and maximum values are included in the range. This is the equivalent of "greater than or equal to *<min>* and less than or equal to *<max>*").
- When specifying a range of time values, Synapse allows the use of both lower resolution values (e.g., `YYYY/MM/DD`) and wildcard values (e.g., `YYYY/MM*`) for the minimum and/or maximum range values. In some cases, wildcard time syntax may provide a simpler and more intuitive means to specify some time ranges. For example `inet:whois:rec:asof=2018*` is equivalent to `inet:whois:rec:asof*range=('2018/01/01', '2018/12/31 23:59:59.999')`. See the type-specific documentation for *time* types for a detailed discussion of these behaviors.
- When using keywords (such as `now`) or relative values (such as `-1 hour`) to specify a range of times, the first value in the range is calculated relative to the current time and the second value is calculated relative to the first value.
- If you specify a range value that is nonsencical or exclusionary (such as `( 47, 16 )`), Synapse will **not** generate an error and will simply fail to return results. (The expression is syntacticaly correct, but no value is both greater than 47 and less than 16).

Lift by Set Membership (*in=)

The set membership extended comparator (`*in=`) supports lifting nodes whose *<form>* = *<valu>* or *<prop>* = *<pval>* matches any of a set of specified values. The comparator can be used with any type.

Syntax:

<form> [**:** | **.** | **:_** *<prop>*] ***in =** (*<set_1>* **,** *<set_2>* **,** . . .)

<interface> **:** *<prop>* ***in =** (*<set_1>* **,** *<set_2>* **,** . . .)

Examples:

Lift organization names matching any of the specified values:

```
ou:name *in= ( fsb, gru, svr )
```

Lift all IPv4 addresses associated with any of the specified Autonomous System (AS) numbers:

```
inet:ipv4:asn *in= ( 9009, 20473, 44477 )
```

Lift all tags (syn:tag nodes) whose final tag element matches any of the specified string values:

```
syn:tag:base *in= ( plugx, korplug, sogu, kaba )
```

Lift by Proximity (*near=)

The proximity extended comparator (`*near=`) supports lifting nodes by "nearness" to another node. Currently, `*near=` supports proximity based on geospatial location (i.e., nodes within a given radius of a specified latitude / longitude).

Syntax:

<form> **:** | **.** | **:_** *<prop>* ***near = ((** *<lat>* **,** *<long>* **),** *<radius>*)

Examples:

Lift all locations (geo:place nodes) within 500 meters of the Russian Cryptographic Museum (where the coordinates 55.83069,37.59781 represent the Museum's location):

```
geo:place:latlong *near= ( (55.83069, 37.59781), 500m )
```

Usage Notes:

- Radius can be specified in the following units. The terms recognized by Storm are listed in parentheses. For example, radius can be specified as 2km or '2 km' or '0.5 meters' or '6 feet' (distance expressions that contain spaces need to be enclosed in single or double quotes):
  - Kilometers (km / kilometer / kilometers)
  - Meters (m / meter / meters)
  - Centimeters (cm / centimeter / centimeters)
  - Millimeters (mm / millimeter / millimeters)
  - Miles (mile / miles)
  - Yards (yard / yards)
  - Feet (foot / feet)
- Radius values of less than 1 must be specified with a leading zero (e.g., 0.5km).

- The `*near=` comparator works for geospatial data by lifting nodes within a square bounding box centered at *<lat>,<long>*, then filters the nodes returned by ensuring that they are within the great-circle distance given by the *<radius>* argument.

Lift by (Arrays) (*[ ])

Storm uses a special syntax to lift (or filter) by comparison with one or more elements of an *array* type. The syntax consists of an asterisk (`*`) preceding a set of square brackets (`[ ]`), where the square brackets contain a comparison operator and a value that can match one or more elements in the array. This allows users to match values in the array list without needing to know the exact order or values of the array itself.

Syntax:

<form> `:|.|:_` *<prop>* **[** *<operator>* *<pval>* **]**

Examples:

Lift the x509 certificates (crypto:x509:cert nodes) that reference FQDNs ending with ".xyz":

```
crypto:x509:cert:identities:fqdns *[ = '*.xyz' ]
```

Lift the MITRE ATT&CK groups whose names include the string 'bear':

```
it:mitre:attack:group:names *[ ~= bear ]
```

Usage Notes:

- The comparison operator used must be valid for lift operations for the type used in the array. For example, *inet:fqdn* suffix matching (i.e., `crypto:x509:cert:identities:fqdns *[ = '*.com' ]`), can be used when lifting arrays consisting of domains, but the prefix operator (`^=`), which is only valid when **filtering** `inet:fqdns`, cannot.
- The standard equals (`=`) operator can be used to filter nodes based on array properties, but the value specified must **exactly match** the **full** property value in question. For example:

  `ou:org:names=( "the vertex project", "the vertex project llc", vertex )`
- See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* document for additional details.

Tag Lifts

Tags in Synapse can represent observations or assessments. They are used to provide context to nodes (in the form of "labels" applied to nodes) and to group related nodes.

Storm supports lifting nodes based on the tag(s) applied to the node, as well lifting based on tag timestamps, tag properties, or tag property values.

The "hashtag" symbol (`#`) is used to specify a tag name when lifting by tag.

Lift by Tag

A "lift by tag" operation lifts **all** nodes that have the specified tag.

Syntax:

*<tag>*

Examples:

Lift all nodes that ESET associates with Sednit:

```
#rep.eset.sednit
```

Lift all nodes associated with anonymized infrastructure:

```
#cno.infra.anon
```

Tip: Tags are hierarchical, and each tag element is its own tag; the tag `#cno.infra.anon` consists of the tags `#cno`, `#cno.infra`, and `#cno.infra.anon`. Lifting nodes using a tag "higher up" in the tag hierarchy will lift all nodes with specified tag or any tag "lower down" in the hierarchy. In other words, lifting by `#cno.infra.anon` will lift "anonymized" infrastructure, whether the infrastructure is a VPN (`#cno.infra.anon.vpn`), a TOR node (`#cno.infra.anon.tor`), or an anonymous proxy (`#cno.infra.anon.proxy`).

Lift Form by Tag

Lift form by tag lifts only those nodes of the specified form that have a particular tag.

Syntax:

<form> **#** *<tag>*

Examples:

Lift the FQDN nodes that ESET associates with Sednit:

```
inet:fqdn#rep.eset.sednit
```

Lift the inet:ipv4 nodes associated with DNS sinkhole infrastructure:

```
inet:ipv4#cno.infra.dns.sink.hole
```

Tip: A "lift form by tag" operation is equivalent to a "lift and filter" operation that first lifts **all** nodes of the specified form, and then filters the results to **only** those nodes with the specified tag. This set of operations is potentially resource-intensive and inefficient (why lift **all** nodes only to discard most of them?). Instead, Synapse specifically optimizes "lift form by tag" operations to **only** lift nodes that have the tag.

In fact, if you specify a Storm query such as `inet:fqdn +#rep.mandiant.apt1`, Synapse will execute the query as if you had specified the "lift form by tag" query `inet:fqdn#rep.mandiant.apt1`. In other words, in some cases Synapse knows to "do what you mean" in order to process your queries more efficiently.

Lift Using Tag Timestamps

A tag timestamp can be thought of as a specialized "property" of a tag that happens to be a date / time range (interval). You can lift nodes based on tag timestamp values using any comparison operator supported by interval (*ival* types). The time / interval extended operator (`@=`) is used most often, but equal to (`=`) can also be used to **exact** match the values in the interval.

See *Lift by Time or Interval (@=)* for additional detail on the use of the `@=` operator.

Syntax:

[*<form>*] **#** *<tag>* **@=** *<time>* **|** **(** *<min_time>* **,** *<max_time>* **)**

Lift any nodes that were associated with anonymous VPN infrastructure between December 1, 2023 and January 1, 2024:

```
#cno.infra.anon.vpn @= ( 2023/12/01, 2024/01/01 )
```

Lift the FQDNs that were owned / controlled by Threat Cluster 15 as of October 30, 2021:

```
inet:fqdn#cno.threat.t15.own @= 2021/10/30
```

Lift the IP addresses that were TOR exit nodes between April 1, 2023 and July 1, 2023:

```
inet:ipv4#cno.infra.anon.tor.exit @= ( 2023/04/01, 2023/07/01 )
```

Lift Using Tag Properties

Tag Properties can be used to provide additional context to tags. Storm supports lifting nodes whose tags have a specific tag property (regardless of the value of the property).

Note: In many cases, information previously recorded using a tag property is better suited to the use of an *Extended Property*.

Syntax:

[*<form>*] **#** *<tag>* **:** *<tagprop>*

Lift any nodes with a ":risk" property reported by Symantec:

```
#rep.symantec:risk
```

Lift all FQDNs with a ":risk" property reported by Symantec:

```
inet:fqdn#rep.symantec:risk
```

Note: You must specify a tag associated with the tag property. It is not possible to lift nodes based on a particular tag property being present on **any** tag (e.g., Storm queries such as `#:risk` or `inet:fqdn#:risk` will generate `BadSyntax` errors).

Lift Using Tag Property Values

Storm supports lifting nodes based on the value of a tag property (similar to lifting by the value of a node property).

You can lift nodes based on tag property values using any comparison operator supported by the property's *Type*. For example, if the tag property is defined as an integer (`int`) type, you can use any comparison operator supported by integers.

The *"Try" Operator* (`?=`) can optionally be used in place of the standard equal to operator (`=`) for tag property values. Use of the try operator is generally not required for interactive Storm queries, but may be useful for more complex Storm queries (such as automation or Storm-based ingest queries).

Syntax:

[*<form>*] **#** *<tag>* **:** *<tagprop>* *<operator>* *<pval>*

Lift any nodes with a ":risk" property value of 100 as reported by ESET:

```
#rep.eset:risk = 100
```

Lift all FQDNs with a ":risk" property value greater than 90 as reported by domaintools:

```
inet:fqdn#rep.domaintools:risk > 90
```

Lift all FQDNs with a ":risk" property with a value between 45 and 70 as reported by Symantec:

```
inet:fqdn#rep.symantec:risk *range= ( 45, 70 )
```

Recursive Tag Lift (##)

Tags can be applied to `syn:tag` nodes to record additional information about the the observation represented by the `syn:tag` node itself. In other words, tags (labels) can be used to provide additional context about tags (`syn:tag` nodes).

The ability to "tag the tags" can be used to represent certain types of analytical relationships. For example:

- `syn:tag` nodes representing threat groups can be tagged to indicate their assessed country of origin.
- `syn:tag` nodes representing malware or tools can be tagged with their assessed availability (e.g., public, private, private but shared, etc.)

A recursive tag lift retrieves all nodes with the specified tag. If the results include any `syn:tag` nodes, the recursive lift will also lift any nodes with those tags. The process continues until no more `syn:tag` nodes are returned.

The final result set returned by a recursive tag lift includes all of the nodes that were lifted recursively, but will **not** include any lifted `syn:tag` nodes themselves.

The "double hashtag" symbol (`##`) is used to specify a recursive tag lift.

Syntax:

*<tag>*

Example:

You are using "availability" tags to show the general availability of malware or tools reported by Mandiant. You add the appropriate "availability" tag to the `syn:tag` node that represents the associated malware. For example, you apply the tag `#rep.mandiant.avail.public` to the node `syn:tag=rep.mandiant.gh0st` because Mandiant reported that the source code for the Gh0st backdoor is publicly available.

Lift all the nodes (e.g., indicators of compromise) associated with any malware family or tool that Mandiant reports is publicly available:

```
##rep.mandiant.avail.public
```

The query above will:

- Lift all nodes tagged `#rep.mandiant.avail.public`, such as `syn:tag` nodes for tools or malware families that Mandiant assesses are publicly available (e.g., `syn:tag=rep.mandiant.gh0st` or `syn:tag=rep.mandiant.beacon`).
- Lift any nodes tagged with those tags (e.g., `#rep.mandiant.gh0st` or `#rep.mandiant.beacon`). This would typically include IOCs such as hashes, FQDNs, IPv4s, URLs, etc.
- If any nodes tagged with the additional tags (`#rep.mandiant.gh0st`, etc.) are `syn:tag` nodes, repeat the process, continuing until no more `syn:tag` nodes are lifted.
- Return the recursively lifted set of nodes (excluding any `syn:tag` nodes).

Note: "Tag the tags" is one approach to provide context to things that tags represent and may be suitable for some use cases. However, the Synapse data model now includes **forms** to represent objects or concepts that are commonly associated with tags, and that can be linked to their associated tag via a `:tag` secondary property. For example, a `risk:tool:software` node can represent Mandiant's reporting on Gh0st malware, with a `:tag` property that could be set to `rep.mandiant.gh0st`. The node can be used to record additional context about Mandiant's Gh0st, including things like availability, alternate names used in reporting, and so on. In short, using a form with secondary properties to provide "context" (that is still linked to an associated tag) gives you greater flexibility to record that context and simplifies lifting, filtering, and pivoting across similar nodes.

See *Tags Associated with Nodes* for a brief discussion of this concept, or the User Guide for the Vertex-Threat-Intel Power-Up (in particular, the Threat Intel Model section) for additional examples.

"reverse" Keyword

Synapse indexes property values so that data (nodes) can be lifted (retrieved) and returned quickly. By default, lift results are returned in lexical order (i.e., sorted in ascending order), based on the property specified in the lift (primary, secondary, universal, or extended) and the way the property is indexed.

The `reverse` keyword can be used to return the specified nodes in reverse lexical order (i.e., sorted in descending order). To perform a "reverse" lift, specify the `reverse` keyword and enclose the lift operation in parentheses.

A "reverse" lift can be followed by additional Storm operations (pivots, filters, commands) just like a "normal" lift.

Tip: When using the `reverse` keyword to lift by secondary property value using an *Interface* name, Synapse performs the lifts for each form in parallel, and yields the results in descending order. For example, the following query will return all nodes of all forms that inherit the `it:host:activity` interface that have a `:time` value greater than or equal to 2024/02/01, sorted in descending order (most recent first):

```
reverse (it:host:activity:time >= 2024/02/01)
```

Syntax:

reverse (*<lift>* **)**

Examples:

Lift inet:ipv4 nodes with a :loc property (sorted descending based on the :loc property value):

```
storm> reverse ( inet:ipv4:loc )
inet:ipv4=197.155.229.194
        :loc = zw.ha.harare
        :type = unicast
        .created = 2024/04/19 14:28:12.123
inet:ipv4=41.221.147.14
        :loc = zw
        :type = unicast
        .created = 2024/04/19 14:28:12.129
inet:ipv4=41.164.23.42
        :loc = za.wc.worcester
        :type = unicast
        .created = 2024/04/19 14:28:09.488
inet:ipv4=155.254.9.3
        :loc = us.mt.three forks
        :type = unicast
        .created = 2024/04/19 14:28:09.494
inet:ipv4=102.64.66.222
        :loc = tz.02.dar es salaam
        :type = unicast
        .created = 2024/04/19 14:28:09.500
```

Lift five inet:ipv4 nodes (sorted descending based on the integer value of the inet:ipv4 primary property):

```
storm> reverse ( inet:ipv4 ) | limit 5
inet:ipv4=255.255.255.255
        :type = private
        .created = 2024/04/19 14:28:12.194
inet:ipv4=223.159.33.195
        :type = unicast
        .created = 2024/04/19 14:28:12.201
inet:ipv4=198.42.76.23
        :type = unicast
        .created = 2024/04/19 14:28:12.207
inet:ipv4=197.155.229.194
        :loc = zw.ha.harare
        :type = unicast
        .created = 2024/04/19 14:28:12.123
inet:ipv4=185.29.8.215
        :type = unicast
        .created = 2024/04/19 14:28:11.761
        #cno.infra.anon.tor.exit = (2023/05/08 14:30:51.000, 2024/01/04 22:05:03.000)
```

Lift the five most recently-created inet:email nodes (sorted descending by the .created property value):

```
storm> reverse ( inet:email.created ) | limit 5
inet:email=illia.volochii@gmail.com
        :fqdn = gmail.com
        :user = illia.volochii
        .created = 2024/04/19 14:28:12.330
inet:email=dholth@fastmail.fm
        :fqdn = fastmail.fm
        :user = dholth
```

(continues on next page)

(continued from previous page)

```
        .created = 2024/04/19 14:28:12.323
inet:email=alex.gronholm@nextday.fi
        :fqdn = nextday.fi
        :user = alex.gronholm
        .created = 2024/04/19 14:28:12.315
inet:email=support@hammer-software.com
        :fqdn = hammer-software.com
        :user = support
        .created = 2024/04/19 14:28:12.309
inet:email=20231128124623.11d85d83ed11a341@adnoc.ae
        :fqdn = adnoc.ae
        :user = 20231128124623.11d85d83ed11a341
        .created = 2024/04/19 14:28:12.303
```

Note: In some cases, Synapse uses specialized indexing to optimize specific Storm operations (such as the ability to lift forms by tag) or to make it easier to work with certain types of data (type-specific behavior). For example, FQDN strings (`inet:fqdn` types) are reversed before being indexed.

Where specialized indexing is used, both "normal" and "reverse" lifts still return nodes in lexical or reverse lexical order, respectively. However, the "sort order" of the results may not be apparent, based on the custom criteria used to index the nodes.

See the *Storm Reference - Type-Specific Storm Behavior* section for details on some type-specific behaviors, including any custom indexing for the listed types.

"Try" Operator

The Storm "try" operator (?=) can be used in lift operations as an alternative to the equal to (=) comparison operator.

Properties in Synapse are subject to *Type Enforcement*. Type enforcement makes a reasonable attempt to ensure that a value "makes sense" for the property in question - that the value you specify for an `inet:ipv4` node looks reasonably like an IPv4 address (and not an FQDN or URL). If you try to lift a set of nodes using a property value that does not pass Synapse's type enforcement validation, Synapse will generate an error. The error will cause the currently executing Storm query to halt and stop processing. For example, the following query halts based on the bad value (`evil.com`) provided for an `inet:ipv4` node:

```
storm> inet:ipv4 = evil.com inet:ipv4 = 8.8.8.8
ERROR: illegal IP address string passed to inet_aton
```

When using the try operator (?=), Synapse will to attempt (try) to lift the node(s) using the specified property value. However, instead of halting in the event of an error, Synapse will ignore the error (silently fail on that specific lift operation) but continue processing the rest of the Storm query. Using the try operator below, Synapse ignores the bad value for the first IPv4 address but returns the second one:

```
storm> inet:ipv4 ?= evil.com inet:ipv4 ?= 8.8.8.8
inet:ipv4=8.8.8.8
        :type = unicast
        .created = 2024/04/19 14:28:11.816
        #rep.domaintools:risk = 42
```

The try operator is generally not necessary for interactive Storm queries. However, it can be very useful for more complex Storm queries or Storm-based automation (see *Storm Reference - Automation*), where a single badly-formatted lift operation (potentially relying on input or data from a third-party data source) could cause the query to fail during execution.

Tip: The try operator can also be used when lifting using an *Interface*.

Syntax:

<form> **?=** *<valu>*

<form> **:** *<prop>* **?=** *<pval>*

<interface> **:** *<prop>* **?=** *<pval>*

Tip: See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* for specialized "try" syntax when working with array properties.

Examples:

Try to lift the MD5 node 174cc541c8d9e1accef73025293923a6:

```
storm> hash:md5 ?= 174cc541c8d9e1accef73025293923a6
hash:md5=174cc541c8d9e1accef73025293923a6
        .created = 2024/04/19 14:28:12.443
```

Try to lift the DNS A nodes whose :ipv4 property is 192.168.0.100:

```
storm> inet:dns:a:ipv4 ?= 192.168.0.100
inet:dns:a=('woot.com', '192.168.0.100')
        :fqdn = woot.com
        :ipv4 = 192.168.0.100
        .created = 2024/04/19 14:28:12.485
```

Try to lift the email address nodes for ron@vertex.link and ozzie@vertex.link:

Notice that despite the first email address being entered incorrectly, the error message is suppressed, and the query executes to completion.

```
storm> inet:email ?= 'ron[at]vertex.link' inet:email ?= 'ozzie@vertex.link'
inet:email=ozzie@vertex.link
        :fqdn = vertex.link
        :user = ozzie
        .created = 2024/04/19 14:28:12.533
```

Try to lift any Microsoft Office document metadata nodes (all nodes of all forms that inherit the file:mime:msoffice interface) whose author (:author property) is 'Rafael Moon':

```
file:mime:msoffice:author ?= 'Rafael Moon'
```

3.6.4 Storm Reference - Filtering

Filter operations are performed on the output of a previous Storm operation such as a lift or pivot. A filter operation downselects from the working set of nodes by either including or excluding a subset of nodes based on specified criteria.

- + specifies an **inclusion** filter. The filter downselects the working set to **only** those nodes that match the specified criteria.
- \- specifies an **exclusion** filter. The filter downselects the working set to all nodes **except** those that match the specified criteria.

Similar to lift operations (*Storm Reference - Lifting*), filter operations can be broken down into "types" of filters based on the criteria, comparison operator, or special handler used:

- *Filter by Form*
- *Filter by Property*
- *Filter by Property Value - Standard Comparison Operators*
- *Filter by Property Value - Extended Comparison Operators*
- *Tag Filters*

Tip: In general, you can filter using the same criteria and comparison operators used for lift operations. This includes using a wildcard (`*`) to partially match form names and using *Interface* names to filter by all forms that inherit an interface.

Because filter operations act on a pre-selected **subset** of nodes, some additional methods are available for filtering that would be less efficient for initial lift operations. For example, you can filter FQDNs (`inet:fqdn` nodes) by prefix (`^=`), although you cannot lift FQDNs using that operator. Similarly, you can *Filter by Tag Globs* but you cannot lift using that syntax.

Storm also supports specialized filters and filter operations:

- *Compound Filters*
- *Subquery Filters*
- *Expression Filters*
- *Embedded Property Syntax*

See *Storm Reference - Document Syntax Conventions* for an explanation of the syntax format used below.

See *Storm Reference - Type-Specific Storm Behavior* for details on special syntax or handling for specific data types.

Filter by Form

A "filter by form" operation modifies your working set to include (or exclude) all nodes of the specified form. The wildcard (asterisk) character (`*`) can be used to filter based on forms that match a partial form name / namespace.

If the form inherits an *Interface*, you can specify the interface name to filter based on all forms that inherit the interface.

Filter by Form Name

Syntax:

<query> **+** | **-** *<form>*

Examples:

Filter the current working set to only include domains (inet:fqdn nodes):

```
<query> +inet:fqdn
```

Filter the current working set to exclude URLs (inet:url nodes):

```
<query> -inet:url
```

Filter by Form Name - Wildcard

You can use the wildcard (asterisk) character (*) to specify all forms that match a partial form name. Use of the wildcard is not limited to form namespace boundaries.

Note: The wildcard can only be used at the end of the partial match. It cannot be used at the beginning or in the middle of the form name. For example, both of the following are **invalid**:

`+*:header`

`-it:exec:*:del`

In addition, use of the wildcard does not extend to partial matching of property names. For example, the following is **invalid**:

`+file:bytes:mime:pe:*`

Syntax:

<query> **+** | **-** *<partial_form_name>* *****

Examples:

Filter the current working set to exclude PE metadata nodes (e.g., file:mime:pe:resource, file:mime:pe:section, etc.):

```
<query> -file:mime:pe:*
```

Filter the current working set to only include antivirus / scan-related nodes (e.g., it:av:scan:result, it:av:signame):

```
<query> +it:av:s*
```

Filter Form by Interface

You can use the name of an interface to filter all forms that inherit that interface.

Note: When filtering by interface, you cannot use the wildcard (*) character to match multiple interface names. Synapse will interpret use of the wildcard as an attempt to match multiple form names.

Syntax:

<query> **+** | **-** *<interface>*

Examples:

Filter the current working set to only include host activity nodes (all nodes of all forms that inherit the it:host:activity interface):

```
<query> +it:host:activity
```

Filter the current working set to exclude taxonomy nodes (all nodes of all forms that inherit the meta:taxonomy interface):

```
<query> -meta:taxonomy
```

Filter by Property

A "filter by property" operation modifies your working set to include (or exclude) all forms that **have** the specified property (secondary, universal, or extended), regardless of the property value.

Tip: When filtering by property, you can specify the property using either the **full** property name (i.e., the combined form and property, such as `inet:dns:a:ipv4`) or the **relative** property name (i.e., the property name alone, including its separator character, such as `:ipv4`).

Using the relative property name allows for simplified syntax and more efficient data entry ("less typing"). Full property names can be used for clarity (i.e., specifying **exactly** what you want to filter on).

Full property names are **required** when filtering on a property using an interface. They may also be required in cases where multiple nodes in the inbound working set have the same relative property name (e.g., `inet:dns:a:ipv4` and `inet:url:ipv4`) and you only wish to filter based on the property of one of the forms.

Each example below is shown using both the full property name (*<form>:<prop>*) and the relative property name (*:<prop>*) where applicable.

Filter by Secondary Property

Syntax:

<query> **+** | **-** [*<form>*] **:** *<prop>*

Examples:

Filter the current working set to only include threats (risk:threat nodes) that have an assessed country of origin (:country:code property):

```
<query> +risk:threat:country:code
```

```
<query> +:country:code
```

Filter the current working set to exclude articles (media:news nodes) that have a publisher name (:publisher:name property):

```
<query> -media:news:publisher:name
```

```
<query> -:publisher:name
```

Filter by Interface Property

If a form inherits an *Interface*, you can filter all nodes of all forms that have an interface-derived property by specifying the full name of the interface and its property.

Syntax:

<query> **+** | **-** *<interface>* **:** *<prop>*

Example:

Filter the current working set to only include those host activity nodes (all nodes of all forms that inherit the it:host:activity interface) that have a :time property:

```
<query> +it:host:activity:time
```

Tip: When filtering using an interface property, you must use full property syntax (i.e., the combined interface and property name).

Filter by Universal Property

Syntax:

<query> **+** | **-** [*<form>*] **.** *<prop>*

Example:

Filter the current working set to only include DNS A records (inet:dns:a nodes) that have a .seen property:

```
<query> +inet:dns:a.seen
```

```
<query> +.seen
```

Filter by Extended Property

Syntax:

<*query*> **+** | **-** [<*form*>] **:_** <*prop*>

Example:

Filter the current working set to exclude those organizations (ou:org nodes) that have an "isthreat" extended property:

```
<query> -ou:org:_vertex:threatintel:isthreat
```

```
<query> -:_vertex:threatintel:isthreat
```

Tip: The `:_vertex:threatintel:isthreat` extended property is a Boolean property added by the Vertex-Threat-Intel Power-Up. It can be used to indicate whether an organization is tracked as a threat group.

Filter by Property Value - Standard Comparison Operators

A "filter by property value" operation modifies the current working set to include (or exclude) the node(s) whose property matches the specified value. This type of filter requires:

- the filter operator (+ or -);
- the property name (full or relative) to use for the filter;
- a *Comparison Operator* to specify how the property value should be evaluated; and
- the property value.

A "filter by property value" can be performed using primary, secondary, universal, or extended properties.

In Synapse, we define **standard comparison operators** as the following set of operators:

- equal to (=)
- less than (<)
- greater than (>)
- less than or equal to (<=)
- greater than or equal to (>=)

For filter operations, the not equal (!=) operator is also supported.

When filtering by secondary or extended property value, you can specify the property using either the **full** property name (i.e., the combined form and property, such as `inet:dns:a:ipv4`) or the **relative** property name (i.e., the property name alone, including its separator character, such as `:ipv4`).

When filtering by universal property value, only the relative property name is required.

Using the relative property name allows for simplified syntax and more efficient data entry ("less typing"). Full property names can be used for clarity (i.e., specifying **exactly** what you want to filter on).

Full property names are **required**:

- when filtering based on an interface property value.

- in cases where multiple nodes in the inbound working set have the same relative property name (e.g., `inet:dns:a:ipv4` and `inet:url:ipv4`, or a universal property such as `.seen`) and you only wish to filter based on the property of one of the forms.

Each example below is shown using both the full property name (*<form>:<prop>*) and the relative property name (*:<prop>*) where applicable.

Tip: When filtering nodes by a property value where the value is a time (date / time), you do not need to use full `YYYY/MM/DD hh:mm:ss.mmm` syntax. Synapse allows you to use either lower resolution values (e.g., `YYYY/MM/DD`) or wildcard values (e.g., `YYYY/MM*`). In particular, wildcard syntax can be used to specify any values that match the wildcard expression. See the type-specific documentation for *time* types for a detailed discussion of these behaviors.

Filter by Primary Property Value

Syntax:

<query> **+** | **-** *<form>* *<operator>* *<valu>*

Filter the current working set to exclude the loopback IPv4 address (127.0.0.1):

```
<query> -inet:ipv4 = 127.0.0.1
```

```
<query> +inet:ipv4 != 127.0.0.1
```

Filter by Secondary Property Value

Syntax:

<query> **+** | **-** [*<form>*] **:** *<prop>* *<operator>* *<pval>*

Filter the current working set to include only those domains (inet:fqdn nodes) that are also logical zones:

```
<query> +inet:fqdn:iszone = 1
```

```
<query> +:iszone  = 1
```

Filter the current working set to exclude any files (file:bytes nodes) with a PE compiled time of 1992-06-19 22:22:17:

```
<query> -file:bytes:mime:pe:compiled = '1992/06/19 22:22:17'
```

```
<query> -:mime:pe:compiled = '1992/06/19 22:22:17'
```

Filter the current working set to include only those files (file:bytes nodes) compiled in 2019:

```
<query> +file:bytes:mime:pe:compiled = 2019*
```

```
<query> +:mime:pe:compiled = 2019*
```

Filter thet current working set to exclude those files (file:bytes nodes) whose size is greater than or equal to 1MB:

```
<query> -file:bytes:size >= 1000000
```

```
<query> -:size >= 1000000
```

Filter by Interface Property Value

If a form inherits an *Interface*, you can filter all nodes of all forms with a specific interface-derived property value by using the name of the interface.

Syntax:

<query> **+** | **-** *<interface>* **:** *<prop>* *<operator>* *<pval>*

Examples:

Filter the current working set to only include those Microsoft Office metadata nodes (all nodes of all forms that inherit the file:mime:msoffice interface) whose :author property is 'admin':

```
<query> +file:mime:msoffice:author = admin
```

Filter the current working set to exclude any host activity nodes (all nodes of all forms that inherit the it:host:activity interface) observed earlier than January 1, 2024:

```
<query> -it:host:activity:time < 2024/01/01
```

Filter by Universal Property Value

Synapse has two built-in universal properties:

- `.created` (a time) which represents the date and time a node was created in Synapse; and
- `.seen` (an interval), a pair of date / time values that can optionally be used to represent when the object represented by a node existed or was observed.

Times (date / time values) are stored as integers (epoch milliseconds) in Synapse and can be filtered using any standard comparison operator.

Because intervals are a pair of date / time values, they can only be filtered using the equal to (=) standard comparison operator to specify an **exact** match for the interval values.

The *Filter by Time or Interval (@=)* and *Filter by Range (*range=)* extended comparison operators provide additional flexibility when filtering by times and intervals.

See the *time* and *ival* sections of the *Storm Reference - Type-Specific Storm Behavior* guide for additional details on working with times and intervals in Synapse.

Syntax:

+ | **-** [*<form>*] **.** *<prop>* *<operator>* *<pval>*

Filter the current working set to include only those nodes created on January 1, 2024 or later:

```
<query> +.created >= 2024/01/01
```

Filter the current working set to include only those FQDNs (inet:fqdn nodes) created on January 1, 2024 or later:

```
<query> +inet:fqdn.created >= 2024/01/01
```

Filter the current working set to include only the DNS A records (inet:dns:a nodes) whose .seen property exactly matches the specified interval:

```
<query> +inet:dns:a.seen = ('2016/06/01 12:22:47.234', '2017/06/10 02:44:55.437')
```

```
<query> +.seen = ('2016/06/01 12:22:47.234', '2017/06/10 02:44:55.437')
```

Filter by Extended Property Value

When filtering by extended property value, you can use any standard comparison operator supported by the property's type. For example, if the extended property is a string, only the equal to (=) standard operator is supported. If the extended property is an integer, any of the standard operators can be used.

Syntax:

+ | **-** [*<form>*] **:_** *<prop>* *<operator>* *<pval>*

Filter the current working set to include only those organizations (ou:org nodes) which are categorized as threats:

```
<query> +ou:org:_vertex:threatintel:isthreat = true
```

```
<query> +:_vertex:threatintel:isthreat = true
```

Tip: Boolean values can be specified using either `true` / `false` or `1` / `0`.

Filter the current working set to incldue only those files whose VirusTotal "reputation" score is less than -100:

```
<query> +file:bytes:_virustotal:reputation < -100
```

```
<query> +:_virustotal:reputation < -100
```

Filter by Property Value - Extended Comparison Operators

Storm supports a set of extended comparison operators (comparators) for specialized filter operations. In most cases, the same extended comparators are available for both lifting and filtering:

- *Filter by Regular Expression (~=)*
- *Filter by Prefix (^=)*
- *Filter by Time or Interval (@=)*
- *Filter by Range (*range=)*
- *Filter by Set Membership (*in=)*
- *Filter by Proximity (*near=)*
- *Filter by (Arrays) (*[ ])*

Each extended comparision operator can be used with any kind of property (primary, secondary, universal, or extended) whose *Type* is appropriate for the comparison used. When filtering by secondary property value, you can optionally specify an *Interface* name and property to filter based on all forms that inherit that interface.

Filter by Regular Expression (~=)

The extended comparator ~= is used to filter nodes based on PCRE-compatible regular expressions.

Tip: *Filter by Prefix (^=)* can be used to filter based on the beginning of string-based properties, and is more efficient for beginning-of-string filter operations. It should be used instead of a regular expression filter where possible.

Syntax:

<query> **+** | **-** *<form>* **~=** *<regex>*

<query> **+** | **-** [*<form>*] **:** | **.** | **:_** *<prop>* **~=** *<regex>*

<query> **+** | **-** *<interface>* **:** *<prop>* **~=** *<regex>*

Examples:

Filter the current working set to include only files (file:bytes nodes) with a PDB path containing the string 'tekide':

```
<query> +file:bytes:mime:pe:pdbpath ~= tekide
```

```
<query> +:mime:pe:pdbpath ~= tekide
```

Filter the current working set to exclude organizations (ou:org nodes) whose name contains a string that starts with "v", followed by 0 or more characters, followed by "x":

```
<query> -ou:org:name ~= '^v.*x'
```

```
<query> -:name ~= '^v.*x'
```

Filter the current working set to only include taxonomy nodes (all nodes of all forms that inherit the meta:taxonomy interface) whose description (:desc property) includes the string 'credential':

```
<query> +meta:taxonxomy:desc ~= credential
```

Filter by Prefix (^=)

Synapse performs prefix indexing on strings and string-derived types, which optimizes filtering nodes whose *<valu>* or *<pval>* starts with a given prefix (substring). The extended comparator ^= is used to filter nodes by prefix.

Note: Extended string types that support dotted notation (such as the *loc* or *syn:tag* types) have custom behaviors with respect to lifting and filtering by prefix.

inet:fqdn nodes are indexed in reverse string order so cannot be **lifted** using the prefix extended operator. However, they can be **filtered** by prefix.

See the relevant sections in the *Storm Reference - Type-Specific Storm Behavior* guide for details.

Syntax:

<query> **+** | **-** *<form>* **^=** *<prefix>*

<query> **+** | **-** [*<form>*] **:** | **.** | **:_** *<prop>* **^=** *<prefix>*

<query> **+** | **-** *<interface>* **:** *<prop>* **^=** *<prefix>*

Examples:

Filter the current working set to exclude email addresses (inet:email nodes) that start with "abuse":

```
<query> -inet:email ^= abuse
```

Filter the current working set to only include organizations (ou:org nodes) whose name starts with "ministry":

```
<query> +ou:org:name ^= ministry
```

```
<query> +:name ^= ministry
```

Filter the current working set to only include Microsoft Office metadata nodes (all nodes of all forms that inherit the file:mime:msoffice interface) whose :author property starts with 'Admin':

```
<query> +file:mime:msoffice:author ^= Admin
```

Filter by Time or Interval (@=)

The time extended comparator (`@=`) is used to filter nodes based on comparisons among various combinations of times and intervals.

Tip: See *Storm Reference - Type-Specific Storm Behavior* for additional detail on the use and behavior of *time* and *ival* data types.

Syntax:

<query> **+** | **-** [*<form>*] **:** | **.** | **:_** *<prop>* **@=(** *<ival_min>* **,** *<ival_max>*)

<query> **+** | **-** [*<form>*] **:** | **.** | **:_** *<prop>* **@=** *<time>*

<query> **+** | **-** *<interface>* **:** *<prop>* **@=(** *<ival_min>* **,** *<ival_max>*)

<query> **+** | **-** *<interface>* **:** *<prop>* **@=** *<time>*

Examples:

Filter the current working set to include only those DNS A records (inet:dns:a nodes) whose .seen values fall between July 1, 2022 and and August 1, 2022:

```
+inet:dns:a.seen @= ( 2022/07/01, 2022/08/01 )
```

```
+.seen @= ( 2022/07/01, 2022/08/01 )
```

Filter the current working set to only include DNS requests (inet:dns:request nodes) that occurred on May 3, 2023 (between 05/03/2023 00:00:00 and 05/03/2023 23:59:59):

```
+inet:dns:request:time @= ( '2023/05/03 00:00:00', '2023/05/04 00:00:00' )
```

```
+:time @= ( '2023/05/03 00:00:00', '2023/05/04 00:00:00' )
```

Tip: Because the `inet:dns:request:time` property is a single date / time value, the following filters would also work:

- `+inet:dns:request:time = 2023/05/03*`

- +:time = 2023/05/03*

Filter the current working set to only include DNS A records (inet:dns:a nodes) whose resolution time window includes the date December 1, 2023:

```
<query> +inet:dns:a.seen @= 2023/12/01
```

```
<query> +.seen @= 2023/12/01
```

Filter results to include only those domain WHOIS records (inet:whois:rec nodes) where the domain was registered (created) exactly on March 19, 2019 at 5:00 UTC:

```
<query> +inet:whois:rec:created @= '2019/03/19 05:00:00'
```

```
<query> +:created @= '2019/03/19 05:00:00'
```

Note: When comparing a single time value to a time property, the @= comparator behaves just like the equal to (=) operator.

Filter the current working set to only include WHOIS email nodes (inet:whois:email) that were observed between July 1, 2023 and the present:

```
<query> +inet:whois:email.seen @= ( 2023/07/01, now )
```

```
<query> +.seen @= ( 2023/07/01, now )
```

Filter the current working set to only include the network flows (inet:flow nodes) that occurred within the past day:

```
<query> +inet:flow:time @= ( now, '-1 day' )
```

```
<query> +:time @= ( now, '-1 day' )
```

Filter the current working set to only include the host activity nodes (all nodes of all forms that inherit the it:host:activity interface) whose :time value is within the past three hours:

```
<query> +it:host:activity:time @= (now, '-3 hours')
```

Usage Notes:

- When specifying an interval with the @= operator, the minimum value is included in the interval for comparison purposes but the maximum value is **not**. This is equivalent to "greater than or equal to <*min*> and less than <*max*>". This behavior differs from that of the *range= operator, which includes **both** the minimum and maximum.
- **Comparing intervals to intervals:** when using an interval with the @= operator to filter nodes based on an interval property, Synapse returns nodes whose interval value has **any** overlap with the specified interval. For example:
  - A lift interval of September 1, 2018 to October 1, 2018 (2018/09/01, 2018/10/01) will match nodes with any of the following intervals:
    * August 12, 2018 to September 6, 2018 (2018/08/12, 2018/09/06).
    * September 13, 2018 to September 17, 2018 (2018/09/13, 2018/09/17).

    * September 30, 2018 to November 5, 2018 (2018/09/30, 2018/11/05).

- **Comparing intervals to times:** When using an interval with the `@=` operator to lift nodes based on a time property, Synapse returns nodes whose time value falls within the specified interval.
- **Comparing times to times:** When using a time with the `@=` operator to filter nodes based on a time property, Synapse returns nodes whose timestamp is an **exact match** of the specified time. In other words, in this case the interval comparator (`@=`) behaves like the equal to comparator (=).
- When specifying date / time and interval values, Synapse allows the use of both lower resolution values (e.g., `YYYY/MM/DD`), and wildcard values (e.g., `YYYY/MM*`). Wildcard time syntax may provide a simpler and more intuitive means to specify some intervals. For example `inet:whois:rec:asof=2018*` is equivalent to `inet:whois:rec:asof@=('2018/01/01', '2019/01/01')`.
- Time-based keywords (such as `now`) and relative time syntax (expressions such as `+-1 hour` or `-7 days`) can be used for interval values.

  See the type-specific documentation for *time* and *ival* types for a detailed discussion of these behaviors.

Filter by Range (*range=)

The range extended comparator (`*range=`) supports filtering nodes whose *<form>* = *<valu>* or *<prop>* = *<pval>* fall within a specified range of values. The comparator can be used with types such as integers and times.

Note: The `*range=` operator can be used to filter both `inet:ipv4` and `inet:ipv6` values (which are stored as decimal integers and strings, respectively). However, ranges of *inet:ipv4* and inet:ipv6 nodes can also be filtered directly by specifying the lower and upper addresses in the range using `<min>-<max>` format. For example:

- `+inet:ipv4 = 192.168.0.0-192.168.0.10`
- `+:ipv4 = 192.168.0.0-192.168.0.10`

Because IPv6 nodes are stored as strings, the range must be enclosed in quotes:

- `+inet:ipv6 = "::0-ff::ff"`
- `+:ipv6 = "::0-ff::ff"`

The `*range=` operator cannot be used to compare a time range with a property value that is an interval (`ival` type). The interval (`@=`) operator should be used instead.

Syntax:

<query> **+ | -** *<form>* ***range = (** *<range_min>* **,** *<range_max>* **)**

<query> **+ | -** [*<form>*] **: | . | :_** *<prop>* ***range = (** *<range_min>* **,** *<range_max>* **)**

<query> **+ | -** *<interface>* **:** *<prop>* ***range = (** *<range_min>* **,** *<range_max>* **)**

Examples:

Filter the current working set to exclude files (file:bytes nodes) whose size is between 1000 and 100000 bytes:

```
<query> -file:bytes:size *range= ( 1000, 100000 )
```

```
<query> -:size *range= ( 1000, 100000 )
```

Filter the current working set to only include files (file:bytes nodes) whose VirusTotal "reputation" score is between -20 and 20:

```
<query> +file:bytes:_virustotal:reputation *range= ( -20, 20 )
```

```
<query> +:_virustotal:reputation *range= ( -20, 20 )
```

Filter the current working set to exclude domain WHOIS records (inet:whois:rec nodes) that were captured / retrieved between November 29, 2013 and June 14, 2016:

```
<query> -inet:whois:rec:asof *range= ( 2013/11/29, 2016/06/14 )
```

```
<query> -:asof *range= ( 2013/11/29, 2016/06/14 )
```

Filter the current working set to only include DNS requests (inet:dns:request nodes) made within one day of December 1, 2021:

```
<query> +inet:dns:request:time *range= ( 2021/12/01, '+-1 day' )
```

```
<query> +:time *range= ( 2021/12/01, '+-1 day' )
```

Filter the current working set to only include taxonomy nodes (all nodes of all forms that inherit the meta:taxonomy interface) whose :depth is between 1 and 3 (i.e., between 2 and 4 taxonomy elements):

```
<query> +meta:taxonomy:depth *range= (1, 3)
```

Usage Notes:

- When specifying a range (`*range=`), both the minimum and maximum values are **included** in the range (the equivalent of "greater than or equal to <*min*> and less than or equal to <*max*>"). This behavior is slightly different than that for time interval (`@=`), which includes the minimum but not the maximum.
- When specifying a range of time values, Synapse allows you to use either lower resolution values (e.g., `YYYY/MM/DD`) or wildcard values (e.g., `YYYY/MM*`) for the minimum and/or maximum range values. In some cases, plain wildcard time syntax may provide a simpler and more intuitive means to specify some time ranges. For example `+inet:whois:rec:asof=2018*` (or `+:asof=2018*`) is equivalent to `+inet:whois:rec:asof*range=('2018/01/01', '2018/12/31 23:59:59.999')` (or `+:asof*range=('2018/01/01', '2018/12/31 23:59:59.999')`). See the type-specific documentation for *time* types for a detailed discussion of these behaviors.
- When using keywords (such as `now`) or relative values (such as `-1 hour`) to specify a range of times, the first value in the range is calculated relative to the current time and the second value is calculated relative to the first value.
- If you specify a range value that is nonsencical or exclusionary (such as `( 47, 16 )`), Synapse will **not** generate an error and will simply fail to return results. (The expression is syntacticaly correct, but no value is both greater than 47 and less than 16).

Filter by Set Membership (*in=)

The set membership extended comparator (*in=) supports filtering nodes whose *<form>* = *<valu>* or *<prop>* = *<pval>* matches any of a set of specified values. The comparator can be used with any type.

Syntax:

<query> **+ | -** *<form>* ***in = (** *<set_1>* **,** *<set_2>* **, …**)

<query> **+ | -** [*<form>*] **: | . | :_** *<prop>* ***in = (** *<set_1>* **,** *<set_2>* **, …**)

<query> **+ | -** *<interface>* **:** *<prop>* ***in = (** *<set_1>* **,** *<set_2>* **, …**)

Examples:

Filter the current working set to exclude organization names (ou:name nodes) matching any of the specified values:

```
<query> -ou:name *in= ( fsb, gru, svr )
```

Filter the current working set to only include IPv4 addresses associated with any of the specified Autonomous System (AS) numbers:

```
<query> +inet:ipv4:asn *in= ( 9009, 20473, 44477 )
```

```
<query> +:asn *in= ( 9009, 20473, 44477 )
```

Filter the current working set to only include tags (syn:tag nodes) whose final tag element matches any of the specified string values:

```
<query> +syn:tag:base *in= ( plugx, korplug, sogu, kaba )
```

```
<query> +:base *in= ( plugx, korplug, sogu, kaba )
```

Filter by Proximity (*near=)

The proximity extended comparator (*near=) supports filtering nodes by "nearness" to another node. Currently, *near= supports proximity based on geospatial location (i.e., nodes within a given radius of a specified latitude / longitude).

Syntax:

<query> **+ | -** [*<form>*] **: | . | :_** *<prop>* ***near = ((** *<lat>* **,** *<long>* **),** *<radius>*)

Examples:

Filter the current working set to only include locations (geo:place nodes) within 500 meters of the Russian Cryptographic Museum (where the coordinates 55.83069,37.59781 represent the Museum's location):

```
<query> +geo:place:latlong *near= ( (55.83069, 37.59781), 500m )
```

Usage Notes:

- In the example above, the latitude and longitude of the desired location are explicitly specified as parameters to *near=.
- Radius can be specified in the following units. The values in parentheses are the acceptable terms for specifying a given unit:
  - Kilometers (km / kilometer / kilometers)

- Meters (m / meter / meters)
- Centimeters (cm / centimeter / centimeters)
- Millimeters (mm / millimeter / millimeters)
- Miles (mile / miles)
- Yards (yard / yards)
- Feet (foot / feet)

- Radius values of less than 1 must be specified with a leading zero (e.g., 0.5 km).
- The `*near=` comparator works for geospatial data by lifting nodes within a square bounding box centered at *<lat>*,*<long>*, then filters the nodes returned by ensuring that they are within the great-circle distance given by the *<radius>* argument.

Filter by (Arrays) (*[ ])

Storm uses a special syntax to filter (or lift) by comparison with one or more elements of an *array* type. The syntax consists of an asterisk (`*`) preceding a set of square brackets (`[ ]`), where the square brackets contain a comparison operator and a value that can match one or more elements in the array. This allows users to match any value in the array list without needing to know the exact order or values of the array itself.

Note: When filtering based on a value in an array property, you must use the relative name of the property. The full property name (i.e., the combined form and property) is not supported for this type of filter.

Syntax:

<query> **+** | **-** **:** | **.** | **:_** *<prop>* ***[** *<operator>* *<pval>* **]**

Examples:

```
<query> +:identities:fqdns *[ = '*.xyz' ]
```

Filter the current working set to exclude the MITRE ATT&CK groups (it:mitre:attack:group nodes) whose names include the string 'bear':

```
<query> -:names *[ ~= bear ]
```

Usage Notes:

- The comparison operator used must be valid for filter operations for the type used in the array.
- The standard equals (=) operator can be used to filter nodes based on array properties, but the value specified must **exactly match** the **full** property value in question:
  - For example: `ou:org +:names=("the vertex project","the vertex project llc",vertex)` will filter to any `ou:org` nodes whose `:names` property consists of **exactly** those names in **exactly** that order.
- See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* document for additional details on working with arrays.

Tag Filters

Tags in Synapse can represent observations or assessments. They are used to provide context to nodes (in the form of "labels" applied to nodes) and to group related nodes.

Storm supports filtering nodes based on the tags applied to nodes (including the use of tag globs), as well as filtering based on tag timestamps, tag properties, or tag property values.

The "hashtag" symbol (#) is used to specify a tag name when filtering by tag.

Filter by Tag (#)

A "filter by tag" operation downselects the current working set to include (or exclude) all nodes with the specified tag.

Syntax:

<query> **+** | **-** **#** *<tag>*

Filter the current working set to exclude all nodes that ESET associates with Sednit:

```
<query> -#rep.eset.sednit
```

Filter the current working set to only include nodes associated with anonymized infrastructure:

```
<query> +#cno.infra.anon
```

Tip: Tags are hierarchical, and each tag element is its own tag; the tag `#cno.infra.anon` consists of the tags `#cno`, `#cno.infra`, and `#cno.infra.anon`. Filtering nodes using a tag "higher up" in the tag hierarchy will include (or exclude) nodes with the specified tag or any tag "lower down" in the hierarchy. In other words, filtering by `#cno.infra.anon` will filter all "anonymized" infrastructure, whether the infrastructure is a VPN (`#cno.infra.anon.vpn`), a TOR node (`#cno.infra.anon.tor`), or an anonymous proxy (`#cno.infra.anon.proxy`).

Filter by Tag Globs

Synapse supports filtering based on the set of tags that match a specified glob expression using single (*) or double (**) asterisks, or a combination of the two.

The single asterisk and double asterisk behave differently:

- The asterisk (*) represents an arbitrary string that matches **within** a single tag element (i.e., one element as bounded by the tag's "dot" (.) separators).
- The double asterisk (**) represents an arbitrary string match anywhere in the tag, including **across** tag elements.

Another way to look at this is that the single asterisk is constrained by the tag's "dot" boundaries, but the double asterisk is not.

Syntax:

<query> **+** | **-** **#** *<string>* | ***** | ****** [**.** *<string>* | ***** | ****** ...]

Examples:

Filter the current working set to exclude any nodes tagged as "seduploader" by any third-party reporting organization:

```
<query> -#rep.*.seduploader
```

To record assessments made by third parties, The Vertex Project uses `rep` ("reported by") as the root tag element, followed by a tag element for the reporting organization (e.g., `rep.eset`), followed by the name of the "thing" reported (in this case, the `seduploader` malware family).

The tag glob filter above uses the single asterisk to match any tag element in the second position, and will match tags such as:

- `rep.eset.seduploader`
- `rep.paloalto.seduploader`

…etc.

Filter the current working set to include any nodes tagged as "cobaltstrike" by any third-party reporting organization whose name begins with 'm':

```
<query> +#rep.m*.cobaltstrike
```

The tag glob filter above uses the single asterisk to match any partial tag element in the second position that starts with 'm', and will match tags such as:

- `rep.malwarebazaar.cobaltstrike`
- `rep.mandiant.cobaltstrike`
- `rep.microsoft.cobaltstrike`

Tip: The filter above would **not** match on a tag such as `rep.malwarebazaar.3p.anyrun.cobaltstrike`, because the string `cobaltstrike` is not the third tag element. A double asterisk, which matches across a tag's "dot" boundaries, would match this tag as well as the example tags above:

`rep.m**cobaltstrike`

Filter the current working set to exclude any nodes tagged as "seduploader" either internally or by any third-party reporting organization:

```
<query> -#*.*.seduploader
```

The Vertex Project uses the `cno` root tag to represent our own internal assessments (and distinguish them from third party-assessments), and the `mal` element to represent assessments related to malware. The tag glob filter above uses two single asterisks to match any element in both the first and second positions, and will match all of the following:

- `rep.eset.seduploader`
- `rep.paloalto.seduploader`
- `cno.mal.seduploader`

…etc.

Filter the current working set to include any nodes reported by Microsoft whose tags end in "blizzard":

```
<query> +#rep.microsoft.**blizzard
```

The tag glob filter above uses a double asterisk to match any Microsoft tag (tag that begins `rep.microsoft`) that ends in "blizzard", regardless of tag depth. The filter will match all of the following:

- `rep.microsoft.aqua_blizzard`
- `rep.microsoft.cadet_blizzard`
- `rep.microsoft.forest_blizzard`

- rep.microsoft.very.long.tag.thatendswithblizzard

...etc.

Filter the current working set to exclude any nodes tagged with any tag that starts with "cno" and is followed by any string:

```
<query> -#cno**
```

The tag glob filter above uses a double asterisk to match any string following "cno". The filter will match all of the following:

- cno.mal
- cno.threat.t42
- cnoooo.you_get_a_cno.and_you_get_a_cno

...etc.

Note: The double asterisk must match "something" - the filter above will **not** match a node that simply has the tag #cno.

Filter the current working set to include any nodes tagged by any third-party reporting organization where the tag contains the string "2017":

```
<query> +#rep.*.**2017**
```

The tag glob filter above uses both a single and double asterisk. The single asterisk matches any tag element in the second position; the double asterisk matches any string that includes "2017", including across "dot" boundaries. The filter will match all of the following:

- rep.alienvault.cve20178291
- rep.malwarebazaar.3p.reversinglabs.document_ole_exploit_cve_2017_11882
- rep.vt.cve_2017_0199

Note: The double asterisk must match "something" - the filter above matches strings where "2017" appears **between** other arbitrary characters. The filter would not match tags such as rep.foo.2017 or rep.bar.baz.cve2017.

Filter Using Tag Timestamp Values

A tag timestamp can be thought of as a specialized "property" of a tag that happens to be a date / time range (interval). You can filter nodes based on tag timestamp values using any comparison operator supported by interval (*ival* types). The time / interval extended operator (@=) is used most often, but equal to (=) can also be used to **exactly** match the values in the interval.

See *Filter by Time or Interval (@=)* for additional detail on the use of the @= operator.

Syntax:

<query> **+** | **-** **#** *<tag>* **@=** *<time>* | **(** *<min_time>* **,** *<max_time>* **)**

Filter the current result set to only include nodes that were associated with anonymous VPN infrastructure between December 1, 2023 and January 1, 2024:

```
<query> +#cno.infra.anon.vpn @= ( 2023/12/01, 2024/01/01 )
```

Filter the current working set to only include nodes that were owned / controlled by Threat Cluster 15 as of October 30, 2021:

```
<query> +#cno.threat.t15.own @= 2021/10/30
```

Filter Using Tag Properties

Tag Properties can be used to provide additional context to tags. Storm supports filtering nodes whose tags have a specific tag property (regardless of the value of the property).

Note: In many cases, information previously recorded using a tag property is better suited to the use of an *Extended Property*.

Syntax:

<query> **+** | **-** **#** *<tag>* | ***** **:** *<tagprop>*

Filter the current working set to only include nodes with a ":risk" property reported by Symantec:

```
<query> +#rep.symantec:risk
```

Filter the current working set to include nodes with a ":risk" property associated with any tag:

```
<query> +#**:risk
```

Tip: When filtering based on the **existence** of a tag property, you can use tag glob syntax (see *Filter by Tag Globs*) to specify the associated tags. Filters such as `+#rep.*.*:risk` or (per the above example) `+#**:risk` are supported.

When filtering for a specific tag property that appears on **any** tag, either a double asterisk (tag glob, as above) or single asterisk can be used, e.g.: `+#*:risk`. The single asterisk in this instance is not a tag glob, but a special syntax helper for this specific use case. (That is, because entering `+#*:risk` instead of `+#**:risk` is a common user error, Synapse automatically handles this case to "do what you mean".)

Filter Using Tag Property Values

Storm supports filtering nodes based on the value of a tag property (similar to filtering by the value of a node property).

You can filter nodes based on tag property values using any comparison operator supported by the property's *Type*. For example, if the tag property is defined as an integer (`int`) type, you can use any comparison operator supported by integers.

Note: Tag glob syntax (see *Filter by Tag Globs*) is **not** supported when filtering based on a tag property value. For example, a filter such as `+#rep.**:risk>20` will generate a syntax error.

Syntax:

<query> **+** | **-** **#** *<tag>* **:** *<tagprop>* *<operator>* *<pval>*

Filter the current working set to include nodes with a ":risk" property value of 100 as reported by ESET:

```
<query> +#rep.eset:risk = 100
```

Filter the current working set to exclude nodes with a ":risk" property value less than 90 as reported by domaintools:

```
<query> -#rep.domaintools:risk < 90
```

Filter the current working set to include nodes with a ":risk" property with a value between 45 and 70 as reported by Symantec:

```
<query> +#rep.symantec:risk *range= ( 45, 70 )
```

Compound Filters

Storm supports the use of the logical operators **and**, **or**, and **not** (including **and not**) to construct compound filters. You can use parentheses to group portions of the filter statement to indicate order of precedence and clarify logical operations when evaluating the filter.

Note:

- Logical operators must be specified in lower case.
- Synapse evalutes compound filters **in order from left to right**. Depending on the filter, left-to-right order may differ from the standard Boolean order of operations (**not** then **and** then **or**).
- Parentheses should be used to logically group portions of the filter statement if necessary to clarify order of operations.

Syntax:

<*query*> **+** | **-** **(** <*filter*> **and** | **or** | **not** | **and not** . . . **)**

Examples:

Filter the current working set to exclude files (file:bytes nodes) that are less than or equal to 16384 bytes in size and were compiled prior to January 1, 2014:

```
<query> -(file:bytes:size <= 16384 and file:bytes:mime:pe:compiled < 2014/01/01)
```

```
<query> -(:size <= 16384 and :mime:pe:compiled < 2014/01/01)
```

Filter the current working set to only include files (file:bytes nodes) or domains (inet:fqdn nodes) that ESET associates with Sednit:

```
<query> +( ( file:bytes or inet:fqdn ) and #rep.eset.sednit )
```

*Filter the current working set to include only files (file:bytes nodes) and domains (inet:fqdn nodes) that ESET associates with Sednit that are **not** sinkholed:*

```
<query> +( ( file:bytes or inet:fqdn ) and ( #rep.eset.sednit and not #cno.infra.dns.
↪sinkhole ) )
```

Subquery Filters

You can use Storm's subquery syntax (*Storm Reference - Subqueries*) to create filters. A subquery (enclosed in curly braces (`{ }`)) can be placed within a larger Storm query.

Most filter operations in Storm will modify (reduce) your current set of nodes based on some criteria of the **nodes themselves** (e.g., a node's form, property, or tag).

Subquery filters allow you to filter your **current** set of nodes based on some criteria of **nearby** nodes. You use the subquery filter to effectively "look ahead" at nodes one or more pivots away from your current nodes, and filter your current nodes based on the properties of those "nearby" nodes.

When nodes are passed to a subquery filter, they are evaluated against the filter's criteria:

- Nodes are **excluded** ("consumed", discarded) if they evaluate **false.**
- Nodes are **included** (not "consumed", retained) if they evaluate **true.**

The subquery pivot operation (used to "look ahead" at other nodes) is effectively performed in the background (without navigating away from your current working set), which provides a more powerful and efficent way to filter your data. (The alternative would be to **actually** navigate to the nearby nodes, filter those nodes, and then navigate **back** to the data you are interested in.)

You can optionally use a standard (mathematical) comparison operator with a subquery filter, in order to filter your current set of nodes based on the **number of results** returned by executing the subfilter's Storm query.

Refer to the *Storm Reference - Subqueries* guide for additional information on subqueries and subquery filters.

Syntax:

<query> **+** | **-** **{** *<query>* **}** [*<standard operator>* *<value>*]

Examples:

Filter the current working set of FQDNs (inet:fqdn nodes) to only FQDNs that have resolved to an IPv4 address that Trend Micro associates with Pawn Storm (i.e., an IP address tagged #rep.trend.pawnstorm):

```
<inet:fqdn> +{ -> inet:dns:a -> inet:ipv4 +#rep.trend.pawnstorm }
```

The subquery filter above takes the inbound `inet:fqdn` nodes and (within the subquery):

- pivots to the associated DNS A records (`inet:dns:a` nodes);
- pivots to the asssociated IPv4 addresses (`inet:ipv4` nodes);
- checks the IPv4 for the presence of a `#rep.trend.pawnstorm` tag.

The subquery filter returns only those `inet:fqdn` nodes where, if you performed the operations within the subquery, **would** (based on the inclusive filter) result in an `inet:ipv4` node with a `#rep.trend.pawnstorm` tag.

Filter the current working set of IPv4 addresses (inet:ipv4 nodes) to exclude any IPv4 associated with an Autonomous System (AS) whose name starts with "makonix":

```
<inet:ipv4> -{ :asn -> inet:asn +:name ^= makonix }
```

The subquery filter above takes the inbound `inet:ipv4` nodes and (within the subquery):

- pivots to the associated `inet:asn` nodes; and
- checks the `inet:asn` nodes for a `:name` value that starts with "makonix".

The subquery filter returns only those `inet:ipv4` nodes where, if you performed the operations within the subquery, **would not** (based on the exclusive filter) result in an `inet:asn` node with a `:name` value starting with "makonix".

Tip: See *Embedded Property Syntax* for an alternative way to perform this query.

Filter the current working set of files (file:bytes nodes) to include only files that are detected as malicious in (10) or more scans (i.e., files that are associated with 10 or more it:av:scan:result nodes whose :verdict property value is "malicious"):

```
<file:bytes> +{ -> it:av:scan:result +:verdict=malicious }>=10
```

The subquery filter above takes the inbound `file:bytes` nodes and (within the subquery):

- pivots to the associated `it:av:scan:result` nodes; and
- filters the results to include only those nodes whose `it:av:scan:result:verdict` property value is `malicious`; and
- counts the number of resulting `it:av:scan:result` nodes for each file.

The subquery filter returns only those `file:bytes` nodes with 10 or more associated `it:av:scan:result` nodes with a `malicious` verdict.

Tip: This is a simplified example. `it:av:scan:result` nodes represent a scan performed at a given point in time; the filter above does not provide any time constraints so will count any / all "malicious" results, regardless of "when" the scan was performed. Results could include files detected as malicious by ten different vendors during a single scan as well as files detected as malicious by only one vendor during ten different scans.

Filter the current working set of x509 certificates (crypto:x509:cert nodes) to only include certificates linked to more than one FQDN (inet:fqdn) identity:

```
<crypto:x509:cert> +{ :identities:fqdns -> inet:fqdn }>1
```

The subquery filter above takes the inbound `crypto:x509:cert` nodes and (within the subquery):

- uses the `:identities:fqdns` array property to pivot to any associated FQDNs (`inet:fqdn` nodes); and
- counts the number of `inet:fqdn` nodes associated with each certificate.

The subquery filter returns only those `crypto:x509:cert` nodes associated with more than one FQDN.

Tip: See *Expression Filters* below for an alternative way to perform this query.

Expression Filters

An expression filter is used to downselect your current working set based on the evaluation of a particular expression. Expression filters are useful when:

- you need to compute a value that you want to use for the filter, or
- when you want to filter based on a value that may change (e.g., when using Storm queries that assign variables; see *Storm Reference - Advanced - Variables*).

Syntax:

<query> **+ | - $(** *<expression>* **)**

Examples:

Filter the current working set of x509 certificates (crypto:x509:cert nodes) to only include certificates linked to more than one FQDN (inet:fqdn) identity:

```
<crypto:x509:cert> $fqdns=::identities:fqdns +$( $fqdns.size() > 1 )
```

This example assigns the list of domains in the `crypto:x509:cert:identities:fqdns` property to the user-defined variable `$fqdns`, computes the number of domains in the list using *size()*, and checks to see if the result is greater than 1.

(See the *Storm Library Documentation* for additional detail on Storm types and Storm libraries.)

Tip: This certificate example is identical to the final example under *Subquery Filters* above, and shows an alternative way to return the same data.

This expression filter is more efficient than the subquery filter because the expression filter simply evaluates the expression ("what is the size of the `:identities:fqdns` array property?"), where the subquery filter needs to pivot to the adjacent nodes in order to evaluate the results. This difference in performance is negligible for small data sets but more pronounced when working with large numbers of nodes.

Filter the current working set of network flows (inet:flow nodes) to only include flows where the total number of bytes transferred in the flow between the source (inet:flow:src:txbytes) and destination (inet:flow:dst:txbytes) is greater than 100MB (~100,000,000 bytes):

```
<inet:flow> +$( :src:txbytes + :dst:txbytes >=100000000 )
```

Filter the current set of nodes associated with any threat group or threat cluster (e.g., tagged ``#cno.threat.<threat_name>``), to include only those nodes that are attributed to more than one threat (e.g., that have more than one #cno.threat.<threat_name> tag):

```
#cno.threat +$( $node.globtags(cno.threat.*).size() > 1 )
```

This query may identify nodes that are incorrectly attributed to more than one group, or instances where two or more threat clusters overlap (which may indicate that the clusters actually represent a single set of activity).

This example uses the *$node.globtags()* method to select the set of tags on each node that match the specified expression (`cno.threat.*`) and *size()* to count the number of matches.

Filter the current working set of DNS A records (inet:dns:a nodes) to only include those whose .seen interval falls WITHIN the past 30 day window (e.g., where the <min> value of the .seen interval is greater than or equal to the date 30 days in the past:

```
$ival = $lib.cast( ival, ( now, -30 days ) )
( $start, $stop ) = $ival
inet:dns:a.seen @= $ival
( $min, $max ) = .seen
+$( $min >= $start )
```

The interval comparison operator (`@=`) will lift or filter interval properties (such as `.seen`) if the node's interval has **any overlap** with the comparison value. Using current Storm syntax, this means it is not possible to directly lift or filter for interval values that fall **within** the comparison interval value.

The above query uses **variables** (see *Storm Reference - Advanced - Variables*) to calculate the date/time exactly 30 days prior to the current date/time (`$start` and `$stop`) and uses an expression filter to ensure that the <*min*> value of the node's `.seen` property is more recent than "30 days ago".

The query is repeated here with inline comments to note what each line is doing:

```
$ival = $lib.cast( ival, ( now, -30 days ) )  // Set the variable $ival to a pair of␣
↪date/time values specified
                                              // using the keyword "now" and the␣
↪relative value "-30 days".
( $start, $stop ) = $ival                     // Set the variables $start and $stop to␣
↪the individual date/times
                                              // from $ival.
inet:dns:a.seen @= $ival                      // Lift all inet:dns:a nodes whose .seen␣
↪property has any **overlap**
                                              // with the past 30 days.
( $min, $max ) = .seen                        // Set the variables $min and $max to the␣
↪individual date/times of the
                                              // .seen interval
+$( $min >= $start )                          // Use an expression filter to ensure that␣
↪the $min time of the node's
                                              // .seen value is greater than or equal to␣
↪the $start time of "30 days ago".
```

Embedded Property Syntax

Storm includes a shortened syntax consisting of two colons (`::`) that can be used to reference a secondary property of an **adjacent** node. Because the syntax can be used to "pull in" a property or property value from a nearby node, it is known as "embedded property syntax".

Embedded property syntax expresses something that is similar (in concept, though not in practice) to a secondary-to-secondary property pivot (see *Storm Reference - Pivoting*). The syntax expresses navigation:

- From a **secondary property** of a form (such as `inet:ipv4:asn`), to
- The **form** for that secondary property (i.e., `inet:asn`), to
- A **secondary property** (or property value) of that **target form** (such as `inet:asn:name`).

Tip: This process can be repeated to reference properties of forms more than one pivot away.

Despite its similarity to a pivot operation, embedded property syntax is commonly used for:

- **Filter operations** (specifically, as a more concise alternative to certain *Subquery Filters*)
- **Variable assignment** (see *Storm Reference - Advanced - Variables*)
- Defining an *Embed Column* in the Synapse UI (Optic)

Syntax:

<query> [**+** | **-**] **:** *<prop>* **::** *<prop>* [**::** *<prop>* ...]

<query> [**+** | **-**] **:** *<prop>* [**::** *<prop>* ...] **::** *<prop>* *<operator>* *<pval>*

Note: When using embedded property syntax in Storm, the leading colon (before the name of the initial secondary property) is **required** - e.g., `:asn::name`.

When using this syntax in Optic (the Synapse UI) to create an embed column in Tabular display mode, the initial colon should be **omitted** - e.g, `asn::name`. Optic effectively prepends the initial colon for you.

Filter Examples:

The examples below illustrate the use of embedded property syntax in a filter expresssion.

Filter the current working set of IPv4 addresses (inet:ipv4 nodes) to exclude any IPv4 associated with an Autonomous System (AS) whose name starts with "makonix":

```
<inet:ipv4> -:asn::name ^= makonix
```

Tip: This example is an alternative way to return the same data as the second example under *Subquery Filters* above:

```
<inet:ipv4> -{ :asn -> inet:asn +:name ^= makonix }
```

Filter the current working set of sandbox "file add" operations (it:exec:file:add nodes) to only those "add" operations performed by a file that has a PDB path (:mime:pe:pdbpath property):

```
<it:exec:file:add> +:sandbox:file::mime:pe:pdbpath
```

The `:sandbox:file` property of an `it:exec:file:add` node represents the file (`file:bytes` node) that was executed in the sandbox environment. If this filter were written as a subquery filter, the pivot syntax within the subquery would look like this:

```
<it:exec:file:add> +{ :sandbox:file -> file:bytes +:mime:pe:pdbpath }
```

Instead, embedded property syntax is used to represent the pivot from the `:sandbox:file` property of the `it:exec:file:add` node, to the associated `file:bytes` node, to the file's `:mime:pe:pdbpath` property.

Filter the current working set of sandbox "file add" operations (it:exec:file:add nodes) to only those "add" operations performed by a self-extracting RAR file (i.e., a file with a PDB path whose base file name is sfxrar.pdb):

```
<it:exec:file:add> +:sandbox:file::mime:pe:pdbpath::base = sfxrar.pdb
```

This example expands on the previous example to use two instances of embedded property syntax. If this filter were written as a subquery filter, the pivot syntax within the subquery would look like this:

```
<it:exec:file:add> +{ :sandbox:file -> file:bytes :mime:pe:pdbpath -> file:path +:base =␣
↪sfxrar.pdb }
```

Instead, embedded property syntax is used to represent the pivots from:

- the `:sandbox:file` property of the `it:exec:file:add` node, to the associated `file:bytes` node, to its `:mime:pe:pdbpath` property; and
- the `:mime:pe:pdbpath` property of the `file:bytes` node, to the associated `file:path` node, to its `:base` property.

Variable Assignment Example:

Embedded property syntax can also be used when assigning variables (see *Storm Reference - Advanced - Variables*).

Set the variable $name to the name of the Autonomous System (AS) associated with a given IPv4 address:

```
<inet:ipv4> $name=:asn::name
```

This example uses embedded property syntax to pivot from the inbound `inet:ipv4` node, to the ASN (`inet:asn` node) associated with the IPv4's `:asn` property, and assigns the value of the ASN's `:name` property to the variable `$name`.

3.6.5 Storm Reference - Pivoting

Pivot operations are performed on the output of a previous Storm operation such as a lift or filter. A pivot operation navigates from one set of nodes to another based on some relationship between the nodes. This relationship is commonly one of the following:

- The nodes have properties that share the same value.
- The nodes are joined by a light edge.

While all node-to-node navigation is referred to generically as "pivoting", for clarity we make a distinction with respect to Storm:

- a **pivot** operation navigates between nodes that share a **property value**; and
- a **traversal** operation navigates between nodes that are connected by a **light edge**.

Tip: Traversal is also known as a "walk" operation (i.e., you can *traverse* a light edge or *walk* a light edge).

Pivots and traversals each have their own operator (symbol) used to represent the operation. The operations can also be combined ("pivot and traverse") in various ways using additional operators.

Generically, all variations of pivots and traversals (including combined operations) require:

- a set of source nodes;
- an operator (symbol) for the kind of operation to be performed; and
- the target node(s) for the operation.

Pivot and traversal operations both consume nodes - each operation navigates "away" from the source nodes to the target nodes, which become your new result set / working set. *Join Operations* can be used to retain the source nodes and combine (join) them with the target nodes in your result set.

See *Storm Reference - Document Syntax Conventions* for an explanation of the syntax format used below.

See *Storm Reference - Type-Specific Storm Behavior* for details on special syntax or handling for specific data types.

Navigating the Data Model

To navigate the data in Synapse (pivot between properties or traverse light edges), it helps to understand how forms, properties, and types are related, and to be familiar with common light edge conventions and how they are used to connect various forms.

- The Synapse Data Model *Forms* documentation lists:
  - All forms in Synapse, with their associated properties and types.
  - The edges that can be used to link a given form (whether as the source or target of the edge).
- For *Optic* users, Optic's Data Model Explorer allows you to search, cross-reference, and view data model objects. In addition to the information above, Data Model Explorer also lists all of the forms that a form is **referenced by** (i.e., where that form's type is a secondary property on another form).

Using the Wildcard as a Target

In many cases you can use the wildcard or asterisk (*) as the target of a pivot or traversal to represent "all the things" that can be reached by your operation. One use for the wildcard is to "explore" a subset of connected data - for example, if you are not sure what relationships exist among the data in your instance of Synapse, and you want to see "what's connected".

Keep in mind that individual wildcard pivots or traversals (as described below) **only** show you nodes that are connected to your source nodes by the specific operation performed; this is generally **not** "all the nodes" that are connected. (See the examples below for details on individual operations.)

The following Storm expression can be used to show **all** nodes that are connected to your source nodes by **any** type of property pivot or edge traversal relationship:

```
<source_nodes> tee { --> * } { <-- * }
```

The expression uses the Storm *tee* command to perform two operations on the set of source nodes (a *Pivot Out and Traverse* and a *Pivot In and Traverse*, with the wildcard as the target in each case) and return the combined results.

Tip: This query is equivalent to using the Explore button in the Optic UI to navigate.

There are two minor exceptions to this "show me all the connections" query:

- The query will not return connections to edge nodes where the source node is an *Ndef* of an associated edge node. Edge nodes are not commonly used (in many cases, they have been deprecated in favor of light edges). Pivot operations involving edge nodes are described below.
- The query will not return property connections where nodes may have have a common property **value**, but the properties are of different **types** (use of the wildcard to "find" relationships depends on *Type Awareness*).

Pivot Operations

Pivot operations navigate between sets of nodes that have properties that share a common value. Each pivot operation requires:

- the source node(s) for the pivot;
- a pivot operator (such as `->`); and
- the target of the pivot.

Unless otherwise specified, the target (*<target>*) of a pivot can be:

- a form name (e.g., `hash:md5`);
- a partial form name (wildcard match, e.g., `hash:*`);
- a form and property name (e.g., `file:bytes:md5`);
- an *Interface* name (e.g., `it:host:activity`);
- an interface and property name (e.g., `file:mime:msoffice:application`);
- a list of form names (e.g., `( inet:dns:request, inet:flow )`); or
- a wildcard / asterisk (*).

For the specialized use case of *Raw Pivot Syntax*, the target of the pivot is a Storm expression.

Note: You cannot specify property **values** in pivot operations. For example, the following is invalid:

```
inet:fqdn=vertex.link -> inet:dns:a:ipv4=127.0.0.1
```

If you want to pivot to a specific node or subset of nodes, you must navigate to the target forms, and then filter your results based on the property value(s) you are interested in:

```
inet:fqdn=vertex.link -> inet:dns:a +:ipv4=127.0.0.1
```

Depending on the kind of pivot operation, you may need to specify a **source property** for the pivot as well; see the discussion of *Implicit Pivot Syntax* below.

While there are a few specialized use cases (such as pivoting to or from tags), most pivots involve navigating between the following kinds of properties:

- primary to secondary;
- secondary to primary;
- secondary to secondary; or
- primary to primary.

Primary to primary property pivots are a specialized use case that is commonly handled using *Raw Pivot Syntax*.

Tip: In Synapse, these property-to-property relationships are **implicit** relationships. You do not need to explicitly "define" the relationship or "link" the nodes. In fact, if the source and target properties have the same value **and** the same *Type*, Storm can use Synapse's *Type Awareness* to simplify pivot operations and identify relationships among nodes.

Implicit Pivot Syntax

Pivot operations in Storm can always be executed by **explicitly** specifying the source and target properties for the pivot. This is referred to as **explicit pivot syntax** or "explicit syntax" for short.

When researching network infrastructure, a common set of pivots is to navigate from a set of FQDNs (`inet:fqdn` nodes) to their DNS A records (`inet:dns:a` nodes) and then to the IPv4 addresses (`inet:ipv4` nodes) that the A records point to. The following Storm query performs those pivots using **explicit syntax**:

```
inet:fqdn = vertex.link -> inet:dns:a:fqdn :ipv4 -> inet:ipv4
```

The query:

- lifts the FQDN `vertex.link`;
- pivots from the FQDN to any DNS A node with the same FQDN property value (`-> inet:dns:a:fqdn`); and
- pivots from the `:ipv4` property of the `inet:dns:a` nodes to any `inet:ipv4` nodes with the same value (`:ipv4 -> inet:ipv4`).

We explicitly specify `inet:dns:a:fqdn` as the **target** property of our first pivot; and we explicitly specify the `:ipv4` property of the `inet:dns:a` nodes as the **source** property of our second pivot. Explicit syntax tells Storm **exactly** what you want to do; there is no ambiguity in the query or in "how" you want to navigate the data.

Note: When specifying a secondary property as the source of a pivot (such as `:ipv4` above), you must specify the property using relative property syntax (i.e., using the property name alone). If you were to use full property syntax (`inet:dns:a:ipv4`) Synapse would interpret that as a lift operation - i.e., "after you pivot to the DNS A records with an FQDN of vertex.link, then lift all DNS A records that have an IPv4 property, and pivot to ALL of the associated IPv4 nodes".

Explicit syntax is precise, but there is extra work ("more typing") involved to create the query, especially when there is an "obvious" source and / or target for the pivot. In other words, if you are pivoting from an FQDN (`inet:fqdn`) to a DNS A record (`inet:dns:a`), the `:fqdn` property is the only target property that makes sense.

In these cases, you can use **implicit pivot syntax** (implicit syntax) for your Storm query. Implicit syntax takes advantage of Synapse's *Type Awareness* to "know" which properties can be pivoted to (or from), given the source and target forms for the pivot operation. With implicit syntax, you do not need to specify the source or target property in cases where it is self-evident for the forms used.

Using implicit syntax, we can rewrite the above query as follows:

```
inet:fqdn = vertex.link -> inet:dns:a -> inet:ipv4
```

With implicit syntax, we can simply specify the source and target **forms**, and allow Synapse to identify the source and target **properties** using types and type awareness.

Implicit syntax can be used in the following cases where the source and target properties have the **same type** and the **same value**:

- Primary to secondary property pivots.
- Secondary to primary property pivots.

Tip: This includes cases where the secondary property value is the **node definition** (*Ndef*) of the corresponding primary property (these cases are uncommon, but includes forms such as `risk:technique:masquerade`). Note that this does not extend to legacy edge nodes that are composite forms, which have their own optimizations and pivot syntax (see *Edge Node Pivots*, below).

Implicit pivot syntax **cannot** be used in the following cases:

- Primary to primary property pivots (see *Raw Pivot Syntax*).
- Secondary to secondary property pivots.
- Pivots between primary and secondary properties involving tags (i.e., where the type is `syn:tag`). See *Tag Pivots* below.
- Pivots between primary and secondary properties with the same value but of different **types**.
- Pivots between primary and secondary properties where the source or target is ambiguous (e.g., where a form has more than one property of the same type, and you only want to pivot to (or from) one of them).

Tip: The examples below use implicit syntax where possible and may include explicit syntax for completeness. Where implicit syntax cannot be used, only explicit syntax is shown.

Pivot Out

Most pivots are "pivot out" operations (or variations). "Pivot out" refers to the "direction" of the pivot operator symbol: an "arrow" (->) that points "out" from left to right. "Pivot out" operations are so common that they are generally just referred to as "pivots" or "pivot operations".

Pivot out operations require:

- the source node(s) for the pivot;
- the pivot operator (->); and
- the target of the pivot.

The target(s) that are appropriate for a particular pivot out operation depend on the source and target properties and the specific navigation you want to perform.

Primary to Secondary Property Pivot

When pivoting from the primary property of a set of source nodes to target nodes with the same secondary property value (e.g., from a set of FQDNs to their associated DNS A nodes), the target can be:

- a form name or interface name;
- a form or interface name with a property name; or
- a partial form name (wildcard form match).

Implicit syntax can be used for these pivots if the target property is self-evident / unambiguous.

Syntax:

<query> **->** *<form>* [**:** | **.** | **._** *<prop>*]

<query> **->** *<interface>* [**:** *<prop>*]

<query> **->** *<partial_form_name>* *

Examples:

Pivot from a set of FQDNs (inet:fqdn nodes) to their associated DNS requests (inet:dns:request nodes):

```
<inet:fqdn nodes> -> inet:dns:request
```

The query above uses implicit syntax; Synapse recognizes that the logical target property (given source nodes that are FQDNs) is `:query:name:fqdn`. You could optionally use explicit syntax instead:

```
<inet:fqdn nodes> -> inet:dns:request:query:name:fqdn
```

Pivot from a set of IPv4 addresses (inet:ipv4 nodes) to any network flows (inet:flow nodes) associated with the IPs:

```
<inet:ipv4 nodes> -> inet:flow
```

The query above uses implicit syntax. Note that because `inet:flow` nodes have two target properties of type `inet:ipv4` (`:src:ipv4` and `:dst:ipv4`), the result of this query will be all `inet:flow` nodes where the inbound IPv4s are either the source **or** destination IP. If you only want to see flows where the inbound IPv4s are the destination IP (for example), you must use explicit syntax to clarify this:

```
<inet:ipv4 nodes> -> inet:flow:dst:ipv4
```

Pivot from a set of tags (syn:tag nodes) to the threat clusters (risk:threat nodes) represented by those tags:

```
<syn:tag nodes> -> risk:threat:tag
```

The query above uses explicit syntax. Even though the `:tag` property is of type `syn:tag`, `syn:tag` nodes have specialized handling with respect to pivoting in Storm (see *Pivot from Tags* and *Pivot to Tags* below). Because pivots involving `syn:tag` nodes are handled differently by default, you must use explicit syntax when pivoting between tags (`syn:tag` nodes) and properties whose type is `syn:tag`.

Pivot from a set of FQDNs (inet:fqdn nodes) to any associated DNS records (e.g., inet:dns:a, inet:dns:cname, inet:dns:ns, inet:dns:request, etc.):

```
<inet:fqdn nodes> -> inet:dns:*
```

The query above uses the wildcard (*) as a partial match for any form name that starts with `inet:dns:`.

Pivot from a set of FQDNs (inet:fqdn nodes) to any associated DNS A or DNS AAAA records:

```
<inet:fqdn nodes> -> inet:dns:a*
```

The query above uses the wildcard (*) as a partial match for any form name that starts with `inet:dns:a` followed by zero or more characters.

Pivot from a set of files to all host execution nodes (all nodes of all forms that inherit the it:host:activity interface - e.g., it:exec:file:add, it:exec:url, etc.) associated with those files:

```
<file:bytes nodes> -> it:host:activity
```

The query above uses implicit syntax. The `it:host:activity` interface defines two properties of type `file:bytes` for host activity nodes - `:exe` and `:sandbox:file`. The query above will return all activity nodes where the inbound files appear in either property. If you only want to see activity nodes whose `:exe` property (for example) matches the inbound files, you need to use explicit syntax:

```
<file:bytes nodes> -> it:host:activity:exe
```

Pivot from a set of usernames (inet:user nodes) to any file paths (file:path nodes) where the username is an element of the path (a :base value - for example, where a username might appear in a PDB path):

```
<inet:user nodes> -> file:path:base
```

The query above uses explicit syntax. The `inet:user` form has a type of `inet:user`, but the `:base` property of the `file:path` form has a type of `file:base`. Because the properties are two different types, you must use explicit syntax.

Tip: The query above will return the (partial) `file:path` whose final element is the user name (e.g., if your source node is `inet:user=evilhacker`, the query will return `c:\users\evilhacker`). You can use the Storm *tree* command to recursively pivot through the remaining `file:path` elements to obtain the full path containing the username:

```
<inet:user nodes> -> file:path:base tree { -> file:path:dir }
```

Pivot from a set of FQDNs (inet:fqdn nodes) to the "masquerade" technique nodes (risk:technique:masquerade nodes) associated with those FQDNs:

```
<inet:fqdn nodes> -> risk:technique:masquerade
```

The above query uses implicit pivot syntax.

A `risk:technique:masquerade` node represents an object that is purposely crafted to imitate (masquerade as) another object, typically for malicious purposes. The node records the relationship between the "original" object being imitated (`risk:technique:masquerade:target`) and the object that is imitating it (`risk:technique:masquerade:node`). (Both properties are of type "ndef" (node definition), and consist of `( <form>, <valu> )` pairs.)

Because the query above uses implicit syntax, it will return any `risk:technique:masquerade` nodes where the inbound FQDNs are either the `:target` or `:node` value. If (for example) the inbound FQDNs were a set of suspicious FQDNs and you wanted to return only those "masquerade" nodes where the inbound FQDNs were the `:node` masquerading as a valid FQDN, you would need to use explicit syntax:

```
<inet:fqdn nodes> -> risk:technique:masquerade:node
```

Secondary to Primary Property Pivot

When pivoting from a secondary property of a set of source nodes to target nodes with the same primary property (e.g., from a set of DNS A nodes to their associated FQDNs), the target can be:

- a form name;
- a list of form names;
- a wildcard.

Implicit syntax can be used for these pivots if the source property is self-evident / unambiguous.

Tip: Use of the the wildcard (asterisk) character (`*`) is known as a "wildcard pivot out". This pivot navigates from any/all secondary properties on the source nodes to the nodes corresponding to those property values. Contrast this operation with the "wildcard pivot in", described under *Pivot In*.

Syntax:

<query> [**:** | **.** | **:_** *<prop>*] **->** *<form>*

<query> [**:** | **.** | **:_** *<prop>*] **->** **(** *<form_1>* **,** *<form_2>* ... **)**

<query> **->** *****

Note: If you specify a source property for the pivot, you must use the relative property name (i.e., the property name alone, preceded by its separator character).

Examples:

Pivot from a set of DNS A records (inet:dns:a nodes) to their associated FQDNs (inet:fqdn nodes):

```
<inet:dns:a nodes> -> inet:fqdn
```

The query above uses implicit syntax. Synapse recognizes that, given a target form of `inet:fqdn`, the logical source property is the `:fqdn` property of the `inet:dns:a` nodes. You could optionally use explicit syntax:

```
<inet:dns:a nodes> :fqdn -> inet:fqdn
```

Pivot from a set of DNS NS records (inet:dns:ns nodes) to their associated FQDNs (inet:fqdn nodes):

```
<inet:dns:ns nodes> -> inet:fqdn
```

The query above uses implicit syntax. Because `inet:dns:ns` nodes have two properties of type `inet:fqdn` (`:zone` and `:ns`), Synapse will pivot to the FQDNs associated with both values. If you only want to pivot to the FQDNs associated with the name server (NS) FQDNs (for example), you must use explicit syntax:

```
<inet:dns:ns nodes> :ns -> inet:fqdn
```

Pivot from a set of X509 certificate metadata nodes (crypto:x509:cert nodes) to the associated SHA1 fingerprints (hash:sha1 nodes) and to any FQDNs (inet:fqdn nodes) associated with the certificates:

```
<crypto:x509:cert nodes> -> ( hash:sha1, inet:fqdn )
```

Tip: Specifying a list of target forms allows you to perform a more focused pivot (in contrast to pivoting to any / all target forms using a wildcard). The elements of the list must be enclosed in parentheses and separated by commas.

Pivot from a set of X509 certificate metadata nodes (crypto:x509:cert nodes) to any/all nodes associated with any of the certificates' secondary properties:

```
<crypto:x509:cert nodes> -> *
```

The query above is an example of a **wildcard pivot out**. For any secondary properties on the source nodes, the query will return the associated nodes. For example, if the `crypto:x509:cert:identities:ipv4s` property is set, the query will return the associated `inet:ipv4` nodes. A wildcard pivot out is also known as a "refs out" pivot (for "references") because it pivots to the nodes "referenced by" the source nodes' secondary properties.

Secondary to Secondary Property Pivot

When pivoting from a secondary property of a set of source nodes to target nodes with the same secondary property (e.g., from the `:ipv4` property of a set of DNS A nodes to a set of network flow nodes with the same IPv4 as a `:dst:ipv4` property), the target can be:

- a form name and property name;
- a list of form and property names; or
- an interface name and property name.

You must use explicit syntax to specify both the source and target properties.

Syntax:

<query> **:** | **.** | **:_** *<prop>* **->** *<form>* **:** | **.** | **:_** *<prop>*

<query> **:** | **.** | **:_** *<prop>* **->** *<interface>* **:** *<prop>*

Examples:

Pivot from the WHOIS records (inet:whois:rec nodes) for a set of domains to the DNS A records (inet:dns:a nodes) for the same domains:

```
<inet:whois:rec nodes> :fqdn -> inet:dns:a:fqdn
```

Tip: In many cases, secondary to secondary property pivots are a "shortcut" for two pivot operations (secondary to primary and primary to secondary) that perform the equivalent navigation. For example:

```
<inet:whois:rec nodes> -> inet:fqdn -> inet:dns:a
```

Pivot from a set of DNS requests (inet:dns:request nodes) to all host activity nodes (all nodes of all forms that inherit the it:host:activity interface) that share the same file as their :exe property:

```
<inet:dns:request nodes> :exe -> it:host:activity:exe
```

Pivot from a set of DNS A records (inet:dns:a nodes) to any network flows (inet:flow) or service banners (inet:banner) associated with the IPs:

```
<inet:dns:a nodes> -> ( inet:flow:dst:ipv4, inet:banner:server:ipv4 )
```

Pivot In

The pivot in operator is an “arrow” (`<-`) that points “in” from right to left.

Pivot operations navigate between sets of nodes that share a common property value. There is no “direction” to this relationship; logically, pivot operations are functionally equivalent whether the pivot “arrow” points out (left to right) or in (right to left):

- Pivot from a set of FQDNs to their associated DNS A records:

  `<inet:fqdn nodes> -> inet:dns:a`

- Pivot from a set of FQDNs to the DNS A records that reference them:

  `<inet:fqdn nodes> <- inet:dns:a`

Because Storm evaluates operations from left to right, the “pivot out” arrow is generally more intuitive and has been used to implement nearly all pivot operations in Storm (the second example, above, is not supported and will generate a `StormRuntimeError`).

The pivot in operation is a specialized operation that can only be used with the wildcard (`*`) as a target. This **wildcard pivot in** operation navigates from the primary property of the source node(s) to any nodes where that value is a secondary property. A wildcard pivot in is also known as a “refs in” pivot (for “references”) because it pivots out to the nodes that “reference” the source nodes’ primary property. Contrast this operation with the “wildcard pivot out”, described under *Secondary to Primary Property Pivot*.

Syntax:

<*query*> **<- ***

Example:

Pivot from a set of FQDNs (inet:fqdn nodes) to all nodes with a secondary property that references a source FQDN:

```
<inet:fqdn nodes> <- *
```

A wildcard pivot in will return any node with a secondary property value that matches any of the source FQDNs. For example, the above query could return various DNS records (`inet:dns:a`, `inet:dns:mx`), URLs (`inet:url`), email addresses (`inet:email`), articles (`media:news`), and so on.

Raw Pivot Syntax

Raw pivot syntax is a pivot operation where the target of the pivot is expressed as a **Storm query**. The use of raw pivot syntax is uncommon for interactive Storm queries, but is useful when you need greater flexibility in specifying the pivot target vs. using standard property-to-property pivot syntax.

Raw pivot syntax requires:

- the source node(s) for the pivot;
- a pivot operator; and
- the target of the pivot (specified as a Storm query enclosed in curly braces).

Both pivot (->) and pivot and join (-+>) operators can be used with raw pivot syntax. Pivot in (<-) and pivot in and join (<+-) are not supported.

Use cases for raw pivot syntax include primary-to-primary property pivots and pivots where the value of the target property (primary or secondary) is computed from the input node(s) (e.g., using a variable derived from the inbound nodes. See *Storm Reference - Advanced - Variables* for a discussion of using variables in Storm).

Syntax:

<query> *<pivot operator>* **{** *<query>* **}**

Examples:

The following is a simple example to illustrate the concept.

Pivot from a set of strings (it:dev:str nodes) representing domains to the associated FQDNs (inet:fqdn nodes):

```
<it:dev:str nodes> $fqdn = $node.value() -> { inet:fqdn ?= $fqdn }
```

The query above is a primary-to-primary property pivot between forms of different types. The query lifts the `it:dev:str` nodes, defines the variable `$fqdn` as the value of the node, then pivots to the `inet:fqdn` nodes with the same value. The FQDN nodes are lifted using the Storm query; the *"Try" Operator* is used in the event that any of the inbound `it:dev:str` nodes are invalid as FQDNs.

Note that you could create an equivalent Storm query using only lift and filter operations:

```
<it:dev:str nodes> $fqdn = $node.value() inet:fqdn ?= $fqdn -it:dev:str
```

This second query still lifts the `it:dev:str` nodes and sets the `$fqdn` variable, but then lifts the `inet:fqdn` nodes directly and drops (filters out) the original `it:dev:str` nodes.

While both queries return the same result (the `inet:fqdn` nodes), using raw pivot syntax is slightly more efficient because the pivot operation drops the `it:dev:str` nodes for you (pivots consume nodes); there is no need to filter them out at the end, as in the second example. As always, these efficiencies may be trivial for smaller queries but can be significant for larger queries.

Pivot from a set of HTTP referer headers (inet:http:request:header nodes) to any URLs (inet:url nodes) specified as the header value:

```
inet:http:request:header:name = referer $url = :value -> { inet:url ?= $url }
```

The query above illustrates using raw pivot syntax to pivot from a secondary to a primary property where the properties are of different types. The query lifts all `inet:http:request:header` nodes where the `:name` value is `referer`, sets the associated `:value` property (a `str` type) as the variable `$url`, then pivots to any `inet:url` nodes (an `inet:url` type) that have the same value. The *"Try" Operator* operator is used in the event any of the referer values contain bad or malformed data.

Tag Pivots

Tags in Synapse are unique in that they are both nodes and labels that are applied to other nodes. (See the *Analytical Model* document for additional discussion of tags as both nodes and labels.)

Because of tags' unique role, pivot syntax used with tags navigates between **tags as nodes** (`syn:tag` nodes) and **tags as labels** (i.e., nodes that have the tags applied), as opposed to performing standard property-to-property pivots.

Both the pivot out (->) and pivot out and join (-+>) operator are supported for tag pivots. Pivot in (<-) and pivot in and join (<+-) are not supported.

Tip: The custom behavior used with tag pivots may lead to counterintuitive results when attempting to pivot between tags (`syn:tag` nodes) and properties that are `syn:tag` types (such as `risk:threat:tag` or `ou:technique:tag`).

For example, if you attempt to pivot from a `syn:tag` node used to associate nodes with a threat cluster to the `risk:threat` node representing the cluster, the following Storm query will fail to return the expected results:

```
syn:tag=rep.talos.lazarus -> risk:threat
```

The query looks as though it should return the appropriate `risk:threat` node using implicit syntax to navigate to the `risk:threat:tag` property, using type awareness. However, because the default Storm behavior when pivoting from a `syn:tag` node is to pivot to **nodes that have the tag**, the above query is actualy attempting to pivot to any `risk:threat` nodes that are tagged `#rep.talos.lazarus`. The query is syntactically correct, so will not generate an error; but it will not return the expected nodes (and likely will not return any nodes at all).

Explicitly specifying the `:tag` property will return the `risk:threat` node as expected:

```
syn:tag=rep.talos.lazarus -> risk:threat:tag
```

Pivot to Tags

Pivot to tags syntax allows you to pivot from a set of nodes with tags to the set of `syn:tag` nodes representing those tags. Pivot to tag operations require:

- the source node(s) for the pivot;
- a pivot operator; and
- the set of tags (`syn:tag` nodes) that is the target of the pivot.

Pivot to tags uses the "hashtag" symbol (#) to indicate that the target of the pivot is a set of `syn:tag` nodes (by default, the set of all leaf tags on the source nodes). The target expression can be modified to refer to:

- all tag nodes,
- all tag nodes matching a specified pattern, or
- the tag node matching a specific tag.

Pivot to Leaf Tags

The hashtag symbol (#) used by itself as a target refers to the `syn:tag` nodes for all **leaf** tags applied to the source nodes. A leaf tag is the longest / final tag in a tag tree. For example, if a node has the tag `#rep.eset.sednit` (which is comprised of the three tags `#rep`, `#rep.eset`, and `#rep.eset.sednit`), the leaf tag is `#rep.eset.sednit`.

Syntax:

<query> *<pivot_operator>* **#**

Examples:

Pivot from a set of nodes to the tags (syn:tag nodes) for all leaf tags applied to those nodes:

```
<query> -> #
```

Pivot to All Tags

The hashtag symbol can be used with the wildcard (#*) to pivot to all `syn:tag` nodes applied to the source nodes, not just the leaf tags. For example, if a node has the tag `#rep.eset.sednit` (which is comprised of the three tags `#rep`, `#rep.eset`, and `#rep.eset.sednit`), all three tags will be returned.

Syntax:

<query> *<pivot_operator>* **#***

Pivot from a set of nodes to the tags (syn:tag nodes) for all tags applied to those nodes:

```
<query> -> #*
```

Pivot to Tags Matching a Pattern

The single (*) or double (**) asterisk can be used to specify a set of target tags that match the specified tag glob expression. The single asterisk is used to match **within** a single tag element. The double asterisk is used to match **across** tag elements.

Syntax:

<query> *<pivot_operator>* **#** *<string>* | * | ** [**.** *<string>* | * | ** ...]

Pivot from a set of nodes to the tags (syn:tag nodes) associated with any third-party reporting where the third tag element is "bisonal":

```
<query> -> #rep.*.bisonal
```

Vertex uses the `rep` tag tree to indicate third-party reporting / assertions. The query above will match `syn:tag` nodes from the source nodes where the tag has `rep` as the first tag element, has any value as the second element, and has `bisonal` as the third element. This would include tags such as:

- `rep.alienvault.bisonal`
- `rep.malwarebazaar.bisonal`

Pivot from a set of nodes to the tags (syn:tag nodes) representing any third party reporting where the tag ends in the string "bisonal":

```
<query> -> #rep.**bisonal
```

The query above will match any tag that has `rep` as the first element, followed by any number of elements that end in the string "bisonal". This would include the tags noted above, as well as tags such as:

- `rep.malwarebazaar.3p.intezer.bisonal`

Pivot to Specific Tag

Pivot to the `syn:tag` node for a specific tag by specifying the exact tag as the target.

Syntax:

<query> *<pivot operator>* # *<tag>*

Pivot from a set of nodes to the syn:tag node for the tag "cno.ttp.phish.attach":

```
<query> -> #cno.ttp.phish.attach
```

Pivot from Tags

Pivot from tags syntax allows you to pivot from a set of `syn:tag` nodes to the nodes that have those tags.

Pivot from tag operations require:

- the source node(s) for the pivot (`syn:tag` nodes);
- a pivot operator; and
- the target forms for the pivot.

The target for a pivot from tags operation can be:

- a form name;
- a partial form name (wildcard match);
- an interface name;
- a list of form names; or
- a wildcard.

Pivot from tags pivots to nodes that have the exact source tag(s) applied. For example, pivoting from the tag `syn:tag=rep.eset.sednit` will **not** return nodes with only `#rep` or `#rep.eset`. Conversely, pivoting from `syn:tag=rep` will return nodes with **any** tag in the `rep` tag tree (because the `rep` tag is included in all of those trees).

Tip: A pivot from tags operation is similar to a *Lift by Tag* or *Lift Form by Tag* operation. The following Storm queries are equivalent:

`syn:tag=rep.microsoft.forest_blizzard -> *`

`#rep.microsoft.forest_blizzard`

As are these:

`syn:tag=rep.microsoft.forest_blizzard -> ( inet:fqdn, inet:ipv4 )`

`inet:fqdn#rep.microsoft.forest_blizzard inet:ipv4#rep.microsoft.forest_blizzard`

Pivot from tags can be useful when used with *Pivot to Tags*. For example, you can take a set of inbound tagged nodes, use pivot to tags to navigate to some or all of the associated `syn:tag` nodes, and then use pivot from tags to navigate to other nodes that have the same tags.

Syntax:

<syn:tag node(s)> <pivot operator> <form>

<syn:tag node(s)> <pivot operator> <partial_form_name> *****

<syn:tag node(s)> <pivot operator> <interface>

<syn:tag node(s)> <pivot operator> **(** *<form_1>* **,** *<form_2> ...* **)**

<syn:tag node(s)> <pivot operator> *****

Examples:

Pivot from a set of syn:tag nodes to any files (file:bytes nodes) with those tags applied:

```
<syn:tag nodes> -> file:bytes
```

Pivot from a set of syn:tag nodes to any DNS nodes with those tags applied:

```
<syn:tag nodes> -> inet:dns:*
```

The query above would return (for example) `inet:dns:a` nodes, `inet:dns:request` nodes, etc.

Pivot from a set of syn:tag nodes to any host activity nodes (all nodes of all forms that inherit the it:host:activity interface) with those tags applied:

```
<syn:tag nodes> -> it:host:activity
```

Pivot from a set of syn:tag nodes to any IPv4 (inet:ipv4), IPv6 (inet:ipv6), server (inet:server), or network flow (inet:flow) nodes with those tags applied and retain the syn:tag nodes in the results (pivot and join):

```
<syn:tag nodes> -+> ( inet:ipv4, inet:ipv6, inet:server, inet:flow )
```

Pivot from a set of syn:tag nodes to all nodes that have any of the tags applied:

```
<syn:tag nodes> -> *
```

Edge Node Pivots

Edge nodes (also called digraph nodes) are specialized nodes whose purpose is to link two arbitrary nodes in a specific relationship. Edge nodes are composite forms but are unique in that, because the node(s) in the relationship may be arbitrary, the edge node's primary property consists of at least one value that is a *Node Definition* or **ndef** - that is, a `(<form>, <valu>)` pair vs. a standard `<valu>`. (Some edge nodes include a time value as a third element of the primary property if the arbitrary relationship occurred at a specific time.)

This means that pivots to or from edge nodes must account for having a form **and** property value in common between the source and target, not just a property value.

Both the pivot (`->`) and pivot and join (`-+>`) operators are supported for edge pivots. The pivot in operator (`<-`) is supported for specialized cases. The pivot in and join operator (`<+-`) is not supported.

Note: Composite edge nodes are largely legacy elements in the Synapse data model. In many cases, the use of lightweight (light) edges is now preferred over edge nodes. For example, `edge:has` nodes have largely been replaced by `-(has)>` light edges; and the use of `meta:seen` nodes is discouraged in favor of `-(seen)>` light edges.

Edge nodes may be preferred when you need to record additional information about the relationship (edge nodes have properties, and you can apply tags to edge nodes). We recommend choosing one option (edge nodes or light edges) for a given relationship for consistency (i.e., so analysts do not have to query for the same type of relationship in two different ways).

In addition, where there is a need to:

- link two arbitrary nodes in some type of relationship, and
- record additional information about the relationships (i.e., where use of a light edge is not appropriate)

…the current preference is to create a guid form whose secondary properties include one or more ndefs instead of a composite edge node (an example of this kind of form is the `risk:technique:masquerade` form). In particular, use of a guid form with ndef secondary properties greatly simplifies pivoting to or from nodes that use ndefs, eliminating the need for analysts to use or be aware of specialized *Edge Node Pivots*.

By using guid forms, the ndef value(s) are no longer tied to the node's primary property the way they are in a legacy composite edge node. This allows Synapse and Storm to treat pivots between nodes and ndef properties as "standard" primary-to-secondary or secondary-to-primary property pivots (using an optimization similar to that used for type awareness and implicit pivot syntax).

Pivot to Edge Nodes

Pivoting to edge nodes requires:

- the source node(s) for the pivot;
- the pivot operator;
- the target of the pivot.

When pivoting to edge nodes, the target can be:

- a form name (edge form);
- a partial edge form name (wildcard match) - supported for pivot out / pivot out and join only;
- an interface (in cases where an edge form inherits an interface); or
- a form (edge form) and property name.

Tip: Storm uses some optimizations to simplify working with edge nodes.

When pivoting out (`->` or `-+>`) to a set of edge nodes, the target of the pivot is assumed to be the edge nodes whose `n1` value matches the `(<form>, <valu>)` of the inbound nodes. This means that you do not need to specify the target property for an edge pivot **unless** the target is the edge nodes' `n2` property.

As an alternative, the pivot in operator (`<-`) can be used specifically to pivot from the source nodes to the `n2` property of the target edge nodes.

Note that the use of the pivot in operator (`<-`) to pivot to edge nodes' `n1` property is not supported, even if you specify it as the target property. In addition, the target of a pivot in operation cannot be a partial edge form name (wildcard match).

Syntax:

<query> *<pivot operator>* *<target>* [**:n2**]

<query> **<-** *<edge_form>*

Examples:

Note: The examples below are for illustrative purposes to show the Storm syntax used to navigate edge nodes. Based on current modeling best practices, the relationships shown here using edge nodes would typically be represented using light edges or guid forms.

Pivot from a set of articles (media:news nodes) to the edge nodes (edge:refs nodes) showing things that the articles reference (e.g., such as indicators like hashes or FQDNs):

```
<media:news nodes> -> edge:refs
```

Pivot from a set of vulnerabilities (risk:vuln nodes) to the edge nodes (edge:has nodes) showing which nodes have those vulnerabilities (e.g., such as an it:host):

```
<risk:vuln nodes> -> edge:has:n2
```

Because the `risk:vuln` ndefs are the `n2` value of the edge nodes, we must specify `:n2` as the target property. The following pivot in operation will return the same results:

```
<risk:vuln nodes> <- edge:has
```

Pivot from a set of nodes to any edge nodes (e.g., edge:has, edge:refs) where the inbound nodes are the n1 of any of the edge nodes:

```
<query> -> edge:*
```

Pivot from Edge Nodes

Pivoting from edge nodes requires:

- the source node(s) (edge nodes) for the pivot;
- the pivot operator;
- the target of the pivot.

When pivoting from edge nodes, the target can be:

- a form name;
- a partial form name (wildcard match) - supported for pivot out / pivot out and join only;
- an interface; or
- a wildcard.

Tip: Storm uses some optimizations to simplify working with edge nodes.

When pivoting out (-> or -+>) from a set of edge nodes, the source of the pivot is assumed to be the `n2` of the edge nodes. If you want to pivot out from edge nodes' `n1` property, you must specify `:n1` as the source property.

As an alternative, the pivot in operator (<-) can be used to specifically to pivot from the edge nodes' to the `n1` property. The use of the pivot in operator to pivot from edge nodes' `n2` property is not supported, even if you specify it as the source property. In addition, the target of a pivot in operation cannot be a partial form name (wildcard match).

Syntax:

<edge_form> [**:n1**] *<pivot operator>* *<target>*

<edge_form> **<-** *<form>* | *<interface>* | *

Examples:

Note: The examples below are for illustrative purposes to show the Storm syntax used to navigate edge nodes. Based on current modeling best practices, the relationships shown here using edge nodes would typically be represented using light edges.

Pivot from a set of "has" edge nodes (edge:has nodes) to all of the objects the nodes "have":

```
<edge:has nodes> -> *
```

Pivot from a set of "has" edge nodes (edge:has nodes) to all of the objects that "have" things:

```
<edge:has nodes> :n1 -> *
```

You can also use the pivot in operator to pivot from the `:n1` property by default:

```
<edge:has nodes> <- *
```

Pivot from a set of "has" edge nodes to any vulnerabilities that the objects "have":

```
<edge:has nodes> -> risk:vuln
```

Pivot Across Edge Nodes

Because edge nodes represent relationships, analytically we are often more interested in the nodes on "either side" of the edge node than we are in the edge node itself. For this reason, the pivot operators have been optimized to allow for easily navigating "across" these edge nodes.

Pivoting across edge nodes still entails two pivot operations (pivot to edges and pivot from edges, as described above). Like all Storm operations, each type of pivot can be performed independently and combined with other operations (e.g., lift, pivot to edges, filter, etc.)

When you pivot to edge nodes and immediately pivot from edge nodes (navigating "across" the edges), Storm optimizes the pivot syntax to simplify this process.

Specifically:

Two pivot out operators (-> or -+>) can be combined to easily pivot from:

- source nodes to the edge nodes' `:n1` property, and
- the edge nodes; `:n2` property to the target nodes.

```
<source_nodes> -> <edge_nodes> -> <target_nodes>
```

By optimizing which proprty of the edge node is assumed to be the source or target of the pivot operation, Synapse makes it easy to navigate across the edge relationship intuitively from left (`:n1`) to right (`:n2`) without the need to explicitly specify source and target properties.

Similarly, two pivot in operators (`<-` or `<+-`) can be used to pivot from:

- source nodes to the edge nodes' `:n2` property, and
- the edge nodes' `:n1` property to the target nodes.

```
<source_nodes> <- <edge_nodes> <- <target_nodes>
```

This allows you to navigate intuitively across the edge relationship "backwards" from right (`:n2`) to left (`:n1`).

Traversal Operations

Traversal operations are used to navigate between sets of nodes that are linked using a lightweight (light) edge. Traversal is still considered a "pivot" between sets of nodes, but is named differently to distinguish light edge navigation from property-to-property pivot navigation.

Tip: There are no light edges (i.e., specific light edge verbs) defined within Synapse by default. Users can create and define their own according to their needs. However, both the Synapse Data Model *Forms* documentation and the Optic UI's Data Model Explorer list edges used by The Vertex Project by convention which are recommended for use (note that Power-Ups may create these edges).

The Storm *model*, *edges*, and *lift.byverb* commands can be used to work with light edges in Synapse.

Each traversal operation requires:

- the source node(s) for the operation;
- the traversal operator; and
- the target forms for the traversal.

Traversal is also referred to as "walking" - that is, you can traverse a light edge or walk a light edge.

The traversal operator is an "arrow" with embedded parentheses. The parentheses enclose a string (commonly a verb) for the relationship(s) represented by the edge:

- `-(<verb>)>`

`<verb>` can be a single edge, a list of edges, or a wildcard.

Unlike property-to-property relationships, edge relationships have a **direction**. There is a "source" node (`n1`) and a "target" node (`n2`) for the relationship itself; an article (`media:news` node, the `n1`) can reference an indicator such as a hash (`hash:md5` node, the `n2`) but it does not make sense for a hash to "reference" an article.

The pivot operator (`->`) and its variations "point" from left to right (other than a few specialized cases) by convention. In contrast, the traversal operator can "point" in either direction, depending on which nodes (which "side" of the edge relationship) are inbound. Both of the syntaxes below are equally valid:

`<media:news nodes> -(refs)> <hash:md5 nodes>`

`<hash:md5 nodes> <(refs)- <media:news nodes>`

Tip: In Synapse, lightweight edge relationships are **explicit** relationships. The light edges used to connect nodes must be defined within Synapse, and nodes joined by light edges must be explicitly linked.

Unless otherwise specified, the target (*<target>*) of an edge traversal can be:

- a form name (e.g., `hash:md5`);
- a form name, comparison operator, and value (e.g., a primary property value, such as `inet:fqdn~=news`);
- a form and property name (e.g., `inet:url:fqdn`);
- a form and property name, comparison operator, and value (e.g., a secondary property value, such as `risk:tool:software:used@=(2018, 2022)`)
- a partial form name (wildcard match, e.g., `hash:*`);
- an interface name (e.g., `it:host:activity`);
- a list of form names (e.g., `( hash:sha256, file:bytes )`); or
- a wildcard / asterisk (`*`).

Note: The ability to specify a secondary property **and value** is supported in Synapse 2.x.x, but will be removed in 3.x.x. You can obtain the same result by traversing to a form and property name, and then filtering on the value. For example, the following Storm queries are equivalent:

`media:news -(refs)> inet:url:fqdn~=justice.gov`

and

`media:news -(refs)> inet:url:fqdn +:fqdn~=justice.gov`

Traverse a Specific Edge

Specify the name (verb) of the edge you want to traverse to navigate a single edge.

Syntax:

<query> **-(** *<verb>* **)>** *<target>*

<query> **<(** *<verb>* **)-** *<target>*

Examples:

Traverse the "uses" light edge from a threat cluster (risk:threat node) to the tools or malware (risk:tool:software nodes) used by the cluster:

```
<risk:threat> -(uses)> risk:tool:software
```

Traverse the "references" (refs) light edge from an article (media:news node) to all of the nodes "referenced" by the article:

```
<media:news> -(refs)> *
```

Traverse the "ipwhois" light edge from a set of IPv4 addresses (inet:ipv4 nodes) to the network registration / network WHOIS records (inet:whois:iprec nodes) the IPs are associated with:

```
<inet:ipv4 nodes> <(ipwhois)- inet:whois:iprec
```

Traverse the "seen" light edges from a set of DNS A records (inet:dns:a nodes) to the sources (meta:source nodes) that "saw" (observed or provided data for) the record:

```
<inet:dns:a nodes> <(seen)- meta:source
```

Traverse Multiple Edges

Specify a list of edge names (verbs) to traverse multiple edges to their targets.

Syntax:

<query> **-(** (*<verb1>* **,** *<verb2>* [**,** *<verb3>*]) **)>** *<target>*

<query> **<(** (*<verb1>* **,** *<verb2>* [**,** *<verb3>*]) **)-** *<target>*

Tip: Because each light edge represents a different relationship whose targets may vary widely, the wildcard (*) is frequently used as the target when traversing multiple edges (though this is not a requirement).

Example:

Traverse the "references" (refs) and "seen" light edges from an FQDN to any nodes linked via those light edges (i.e., articles (media:news nodes) that reference the FQDN and data sources (meta:source nodes) that "saw" the FQDN):

```
<inet:fqdn> <( ( refs, seen ) )- *
```

Traverse All Edges

Use the wildcard (asterisk) character (*) to traverse any edges present in the specified direction to their targets.

Tip: Using the wildcard to specify any edge name is useful when:

- you want to traverse any / all edges;
- you want to navigate to a specific set of targets (regardless of the particular edges); or
- if you are not familiar with the edges that may be used with your source nodes and simply want to explore any connections that may be present.

Syntax:

<query> **-(** * **)>** *<target>*

<query> **<(** * **)-** *<target>*

Examples:

For a threat cluster (risk:threat node), traverse any light edges linking the cluster to any vulnerabilities (risk:vuln nodes) or victim organizations (ou:org nodes):

```
<risk:threat> -(*)> ( risk:vuln, ou:org )
```

For a vulnerability (risk:vuln node), navigate to any forms that are connected to the vulnerability by any edge:

```
risk:vuln <(*)- *
```

Vertex Project conventions can link `risk:vuln` nodes to things that have the vulnerability (via <(has)- edges) or things that use the vulnerability (via <(uses)- edges - e.g., a threat cluster (`risk:threat`) or a file (`file:bytes`) that makes use of an exploit for the vulnerability, etc.).

Pivot and Traverse Operations

Property pivots and edge traversals can be combined into a single "pivot and traverse" operation to perform "both types" of navigation simultaneously from a set of source nodes. "Pivot and traverse" combines a pivot operation with a traverse all edges operation in the specified direction.

Each pivot and traverse operation requires:

- the source node(s) for the operation;
- the pivot and traverse operator; and
- the target of the operation.

The pivot and traverse operator is a "double arrow" (`-->`) with two dashes (vs. one for a standard pivot). The operator can point "out" or "in", depending on the specific pivot and traverse operation you want to perform.

Because pivot and traverse operations perform "all available" navigation in a given direction, the only valid target for this operation is the wildcard (`*`).

Tip: The combined "pivot and traverse" operators are commonly used to "explore" a subset of connected nodes. Note that the Storm *tee* command can be used to perform concurrent pivot in and traverse / pivot out and traverse operations on an inbound set of nodes:

```
<query> | tee { --> * } { <-- * }
```

This combined operation shows you **all** nodes connected to your source nodes by **any** property or edge. It is equivalent to using the Explore button in the Optic UI.

Pivot Out and Traverse

The pivot out and traverse (walk) light edges operator (`-->`) combines a wildcard pivot out ("refs out") operation (`-> *`) with a wildcard (any / all edges) edge traversal operation (`-(*)> *`).

Syntax:

<*query*> **-->** *

Examples:

Pivot from a set of IP netblock registration records (inet:whois:iprec nodes) to all nodes associated with the records' secondary properties and all nodes linked to the records by light edges:

```
<inet:whois:iprec nodes> --> *
```

Pivot In and Traverse

The pivot in and traverse (walk) light edges operator (`<-- *`) combines a wildcard pivot in ("refs in") operation (`<- *`) with a wildcard (any / all edges) edge traversal operation (`<(*)- *`).

Syntax:

<*query*> **<--** *

Examples:

Pivot from a set of IPv4 addresses (inet:ipv4 nodes) to all nodes that reference the IPs and all nodes linked to the IPs by light edges:

```
<inet:ipv4> <-- *
```

Join Operations

Like most Storm operations, pivots and traversals consume nodes. With both types of operations, you navigate "away" from your source nodes; the nodes that are the target of your operation become your working set / result set.

With join operations, the inbound nodes are retained and combined (joined) with the target nodes in a single result set.

Each join operation requires:

- the source node(s) for the operation;
- the appropriate join operator; and
- the target of the operation.

Join operators are variations on the standard pivot and traversal operators, but include a "plus" sign (+) in the operator to indicate the join. For example, the pivot and join operator is the pivot arrow (->) combined with the plus sign to represent "pivot and join" (-+>).

Pivot and Join

The pivot and join operator (-+>) is the pivot "arrow" with an embedded plus sign (+) to represent joining the source and target nodes. The operator can be used anywhere the standard pivot operator (->) is used, including the wildcard pivot out. Refer to the syntax examples under *Pivot Out* for the various types of pivot operations (e.g., primary to secondary property, secondary to secondary property, etc.)

Examples:

Pivot from a set of organizations (ou:org nodes) to any associated contacts (ps:contact nodes), retaining the organizations in the results:

```
<ou:org nodes> -+> ps:contact
```

Pivot from a set of DNS A records (inet:dns:a nodes) to their associated IPv4 addresses (inet:ipv4 nodes), retaining the DNS A records in the results:

```
<inet:dns:a nodes> -+> inet:ipv4
```

Pivot from a set of domain WHOIS records (inet:whois:rec nodes) to the DNS A records (inet:dns:a nodes) associated with the FQDNs, retaining the WHOIS records in the results:

```
<inet:whois:rec nodes> :fqdn -+> inet:dns:a:fqdn
```

Pivot from all secondary properties of a set of files (file:bytes nodes) to the associated nodes, retaining the files in the results:

```
<file:bytes nodes> -+> *
```

Pivot In and Join

The pivot in and join operator (`<+-`) can be used to perform a wildcard pivot in operation and join the results with the source nodes. Just as with a standard pivot in, the wildcard is the only valid target for this operation.

Syntax:

<query> **<+-** *

Example:

Pivot from a set of files (file:bytes nodes) to any nodes that reference the files, retaining the original files in the results:

```
<file:bytes nodes> <+- *
```

Traverse and Join

The traverse and join operator (`-(<verb>)+>`) is a traversal "arrow" with a plus sign (+) to represent joining the source and target nodes. The operator can be used anywhere the standard (`-(<verb>)>`) traversal operator is used. The operator can be used with a named edge or edges (e.g., `-(refs)+>` or `<+((uses, targets))-`) or with a wildcard to represent any / all edges (e.g., `-(*)+>`).

Syntax:

<query> **-(** *<verb>* **)+>** *<target>*

<query> **<+(** *<verb>* **)-** *<target>*

<query> **-(** (*<verb1>* **,** *<verb2>* [**,** *<verb3>* ...]) **)+>** *<target>*

<query> **<+(** (*<verb1>* **,** *<verb2>* [**,** *<verb3>* ...]) **)-** *<target>*

<query> **-(** * **)+>** *<target>*

<query> **<+(** * **)-** *<target>*

Examples:

Traverse the "refs" light edge from an article (media:news node) and join the article with the FQDNs (inet:fqdn nodes) "referenced" by the article:

```
<media:news> -(refs)+> inet:fqdn
```

Join an article (media:news node) with any/all nodes referenced by the article:

```
<media:news> -(refs)+> *
```

Join a threat cluster (risk:threat node) with any nodes used or targeted by the cluster:

```
<risk:threat> -( (uses, targets) )+> *
```

Traverse and join any/all light edges from a vulnerability (risk:vuln node) to all nodes linked by any light edge:

```
<risk:vuln> <+(*)- *
```

Common edges used with vulnerabilites (`risk:vuln` nodes) include "uses" and "has".

Pivot, Traverse, and Join

The pivot, traverse, and join operator (--+>) combines the "pivot and traverse" double arrow with a plus sign (+) to represent joining the source and target nodes. The operator can be used anywhere the standard pivot and traverse operator (-->) is used. Because combined pivot and traverse operations perform "all available" navigation, the only valid target for this operation is the wildcard (*).

Syntax:

<*query*> **--+>** *

<*query*> **<+--** *

Examples:

Join a set of articles (media:news nodes) with all nodes representing the articles' secondary properties (pivot out) and all nodes linked by any "right-facing" light edge:

```
<media:news> --+> *
```

Join a set of IPv4 addresses (inet:ipv4 nodes) with all nodes that reference the IPs (pivot in) and all nodes linked to the IPs by "left-facing" light edges:

```
<inet:ipv4> <+-- *
```

3.6.6 Storm Reference - Data Modification

Storm can be used to modify data in Synapse by:

- adding or deleting nodes;
- setting, modifying, or deleting properties on nodes;
- adding or deleting light edges; and
- adding or deleting tags from nodes (including tag timestamps or tag properties).

The ability to create or modify data on the fly gives users a powerful degree of flexibility and efficiency.

Warning: The ability to add and modify data directly from Storm is powerful and convenient, but users can inadvertently modify (or even delete) data inappropriately through mistyped syntax, incorrect Storm logic, or premature striking of the "enter" key. While some built-in protections exist within Synapse itself, it is important to remember that **there is no "are you sure?" prompt before a Storm query executes.**

The following best practices will help prevent inadvertent changes to a Cortex:

- Where possible, *Fork a View* and test and perform your changes in the fork. Once you have validated the changes, they can be merged into the parent view; if anything goes wrong, the fork can simply be deleted.
  - Note that in Synapse 2.x, all **delete** operations must be performed in the layer where the data to be deleted exists; that is, if you are working in forked view, you cannot delete data in the underlying view/layer from the fork.
- Use extreme caution when constructing complex Storm queries that may modify (or delete) large numbers of nodes. It is **strongly recommended** that you validate the output of a query by first running the query on its own (without the edit or delete operations) to ensure it returns the expected results (set of nodes) before permanently modifying or deleting those nodes.

- Use the Synapse permissions system to enforce least privilege. Limit users to permissions appropriate for tasks they have been trained for / are responsible for.

Tip: For adding data at scale, we recommend use of the Synapse *csvtool*, the Synapse *feed* utility, the Optic *Ingest Tool*, or the programmatic ingest of data (e.g., using a *Power-Up*).)

See *Storm Reference - Document Syntax Conventions* for an explanation of the syntax format used below.

See *Storm Reference - Type-Specific Storm Behavior* for details on special syntax or handling for specific data types (*Type*).

Edit Mode

To perform an edit operation in Storm, you must enter "edit mode". Edit mode makes use of several conventions to specify what changes should be made and to what data:

- *Edit Brackets*
- *Edit Parentheses*
- *"Try" Operator*
- *Autoadds and Depadds*

Edit Brackets

The use of square brackets (`[ ]`) within a Storm query can be thought of as entering "edit mode" to perform an edit operation. The data in the brackets specifies the changes to be made involving nodes, properties, light edges, and tags. The only exception is deleting nodes, which is done using the Storm *delnode* command.

The square brackets used for the Storm data modification (edit) syntax indicate "perform the enclosed changes" in a generic way. Edit brackets are used to perform any of the following:

- *Add Nodes*
- *Add or Modify Properties*
- *Add or Modify Properties Using Subqueries*
- *Delete Properties*
- *Add Light Edges*
- *Delete Light Edges*
- *Add Tags*
- *Modify Tags*
- *Remove Tags*

All of the above directives can be specified within a single set of brackets (subject to Storm logic and Storm's pipeline behavior).

> **Warning:** It is critical to remember that **the brackets are NOT a boundary that segregates nodes;** the brackets simply indicate the start and end of an edit operation. In other words, the brackets do **NOT** separate "nodes the modifications should apply to" from "nodes they should not apply to". Storm *Operation Chaining* with left-to-right processing order still applies. Editing is simply another Storm operation, so the specified edits will be performed on **ALL nodes inbound to the edit operation** as part of the Storm pipeline, regardless of whether those nodes are within or outside the brackets.
>
> The exception is modifications that are placed within *Edit Parentheses*, which can be used to segregate specific edit operations.

Note: For simplicity, syntax examples below demonstrating how to add nodes, modify properties, etc. only use edit brackets. See *Combining Data Modification Operations* below for examples showing the use of edit brackets with and without edit parentheses.

Edit Parentheses

Storm supports the use of edit parentheses (`( )`) inside of *Edit Brackets*. Edit parentheses ("parens") explicitly limit a set of modifications to a specific node or nodes by enclosing the node(s) and their associated modification(s) within the parentheses. This "overrides" the default behavior for edit brackets, which is that every change specified within the brackets applies to **all nodes inbound to the edit operation.** Edit parens thus allow you to make limited changes "inline" with a more complex Storm query instead of having to use a smaller, separate query to make those changes.

Note that multiple sets of edit parens can be used within a single set of edit brackets; each set of edit parens delimits a separate set of edits.

See *Combining Data Modification Operations* below for examples showing the use of edit brackets with and without edit parentheses.

"Try" Operator

The Storm "try" operator can be used in edit operations when setting properties (`?=`) or adding tags (`+?#`).

Properties in Synapse are subject to *Type Enforcement*. Type enforcement makes a reasonable attempt to ensure that a value "makes sense" for the property in question - that the value you specify for an `inet:ipv4` node looks reasonably like an IPv4 address (and not an FQDN or URL). If you try to set a property value that does not pass Synapse's type enforcement validation, Synapse will generate a `BadTypeValu` error. The error will cause the currently executing Storm query to halt and stop processing.

When using the try operator, Synapse will to attempt (try) to set the property value. With the try operator, instead of halting in the event of a `BadTypeValu` error, Synapse will ignore the error (silently fail on that specific edit operation) but continue processing the rest of the Storm query.

The try operator is especially useful for Storm-based automated ingest of data where the data source may contain bad (improperly typed or poorly formatted) data, where a single badly-formatted entry could cause an ingest query to fail in the middle.

For example:

```
[ inet:ipv4 ?= woot.com ]
```

will silently fail to create an `inet:ipv4` node with the improper value `woot.com`.

In contrast:

```
[ inet:ipv4 = woot.com ]
```

will throw a `BadTypeValu` error and exit.

Tip: See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* for specialized "try" syntax when working with arrays.

Tags and the "Try" Operator

Tags are also nodes (`syn:tag` nodes), and tag values are also subject to type enforcement. As such, the "try" operator can also be used when applying tags:

```
inet:ipv4 = 58.158.177.102 [ +?#cno.infra.dns.sink.hole ]
```

While Synapse automatically normalizes tag elements (e.g., by replacing dash characters (-) or spaces with underscores (_)), some characters (such as ASCII symbols other than the underscore) are not allowed. The "try" operator may be useful when ingesting third-party data or constructing a tag using a *Variable* where the variable may contain unexpected values. For example:

```
inet:ipv4 = 8.8.8.8 [ +?#foo.$tag ]
```

… where `$tag` is a variable representing a tag element derived from the source data.

See the *syn:tag* section of the *Storm Reference - Type-Specific Storm Behavior* for additional detail on tags / `syn:tag` forms.

Autoadds and Depadds

Synapse makes use of two optimization features when adding nodes or setting secondary properties: automatic additions (*Autoadd*) and dependent additions (*Depadd*).

Autoadd is the process where, on node creation, Synapse will automatically set any secondary properties that are derived from a node's primary property. Because these secondary properties are based on the node's primary property (which cannot be changed once set), the secondary properties are read-only.

For example, when creating the email address `inet:email=visi@vertex.link`, Synapse will automatically set the node's secondary properties (the username `inet:email:user=visi` and domain `inet:email:fqdn=vertex.link`).

Depadd is the process where, on setting a node's secondary property value, if that property is of a type that is also a form, Synapse will automatically create the form with the corresponding primary property value. (You can view this as the secondary property "depending on" the existence of a node with the corresponding primary property.)

To use the same example, when creating the email `inet:email=visi@vertex.link` and setting the secondary properties above, Synapse willl also create the associated nodes `inet:user=visi` and `inet:fqdn=vertex.link`.

Autoadd and depadd work together (and recursively) to simplify adding data to Synapse.

Add Nodes

Operation to add the specified node(s) to a Cortex.

Syntax:

[*<form>* **=** | **?=** *<valu>* … **]**

Tip: You can optionally use the *"Try" Operator* (?=) when adding nodes.

Examples:

Create a simple node (FQDN):

```
[ inet:fqdn = woot.com ]
```

Create a composite (comp) node (DNS A record):

```
[ inet:dns:a=(woot.com, 12.34.56.78) ]
```

Create a GUID node by generating an arbitrary guid using the asterisk character:

```
[ risk:threat='*' ]
```

Create a GUID node by specifying a list of string values used to generate a predictable guid:

```
[ risk:threat=(mandiant, apt1) ]
```

Tip: For information on the differences and use cases for arbitrary vs. predictable / deconflictable guids, see the *guid* section of the *Storm Reference - Type-Specific Storm Behavior*.

Storm also includes various *gen* ("generate") commands to simplify the creation of some common guid forms.

Create a digraph (edge) node:

```
[ edge:refs=((media:news, 00a1f0d928e25729b9e86e2d08c127ce), (inet:fqdn, woot.com)) ]
```

Note: In many cases, the use of an *Edge (Digraph) Form* has been replaced by a *Lightweight (Light) Edge*.

Create multiple nodes in a single edit operation:

```
[ inet:fqdn=woot.com inet:ipv4=12.34.56.78 hash:md5=d41d8cd98f00b204e9800998ecf8427e ]
```

Usage Notes:

- If a node specified within the edit brackets does not exist, Synapse creates and returns the node. If the node already exists, Synapse simply returns (lifts) the node.
- When creating a *<form>* whose *<valu>* consists of multiple components, the components must be passed as a comma-separated list enclosed in parentheses.
- Once a node is created, its primary property (*<form>* = *<valu>*) **cannot be modified.** The only way to "change" a node's primary property is to create a new node (and optionally delete the old node).

Add or Modify Properties

Operation to add (set) or change one or more properties on the specified node(s).

The same syntax is used to apply a new property or modify an existing property.

Syntax:

<query> **[:** *<prop>* **=** | **?=** *<pval>* ... **]**

Tip: You can optionally use the *"Try" Operator* (?=) when setting or modifying properties.

Note: Synapse supports secondary properties that are **arrays** (lists or sets of typed forms), such as `ou:org:names`. See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* guide for the syntax used to add or modify array properties.

Examples:

Add (or modify) a secondary property:

```
<inet:ipv4> [ :loc=us.oh.wilmington ]
```

Add (or modify) a universal property:

```
<inet:dns:a> [ .seen=("2017/08/01 01:23", "2017/08/01 04:56") ]
```

Usage Notes:

- Specifying a property will set the *<prop>* = *<pval>* if it does not exist, or modify (overwrite) the *<prop>* = *<pval>* if it already exists. **There is no prompt to confirm overwriting of an existing property.**
- Storm will return an error if the inbound set of nodes contains any forms for which *<prop>* is not a valid property. For example, attempting to set a `:loc` property when the inbound nodes contain both domains and IP addresses will return an error as `:loc` is not a valid secondary property for a domain (`inet:fqdn`).
- Properties to be set or modified **must** be specified by their relative property name. For example, for the form `foo:bar` with the property `baz` (i.e., `foo:bar:baz`) the relative property name is specified as `:baz`.

Add or Modify Properties Using Subqueries

Secondary property values can be set using a **subquery** to assign the value. The subquery executes a Storm query to lift the node(s) whose primary property should be assigned as the value of the secondary property.

This is a specialized use case that is most useful when working with property values that are guids (see *GUID*) as it avoids the need to type or copy and paste the guid value. Using a subquery allows you to reference the guid node using a more "human friendly" method.

(See *Storm Reference - Subqueries* for additional detail on subqueries.)

Tip: You can optionally use the *"Try" Operator* (?=) when setting or modifying properties using a subquery.

Syntax:

<query> **[:** *<prop>* **=** | **?= {** *<query>* **}** ... **]**

Examples:

Use a subquery to assign an organization's (ou:org) guid as the secondary property of a ps:contact node:

```
storm> ps:contact:orgname="U.S. Department of Justice" [ :org={ ou:org:alias=usgovdoj } ]
ps:contact=d41d8cd98f00b204e9800998ecf8427e
        :address = 950 pennsylvania avenue nw, washington, dc, 20530-0001
        :loc = us.dc.washington
        :org = 0fa690c06970d2d2ae74e43a18f46c2a
        :orgname = u.s. department of justice
        :phone = +1 (202) 514-2000
        .created = 2024/04/19 14:27:51.633
```

In the example above, the subquery `ou:org:alias=usgovdoj` is used to lift the organization node with that `:alias` property value and assign the `ou:org` node's guid value to the `:org` property of the `ps:contact` node.

Use a subquery to assign one or more industries (ou:industry) to an organization (ou:org):

```
storm> ou:org:name=apple [ :industries+={ ou:industry:name="computers and electronics"␣
↪ou:industry:name="telecommunications" } ]
ou:org=2848b564bf1e68563e3fea4ce27299f3
        :alias = apple
        :industries = ['42c577d56d3191a98be1b5bd9f4db8e9',
↪'6ed2f6db1fd3864823a044cb15af265a']
        :loc = us.ca.cupertino
        :name = apple
        :names = ['apple', 'apple, inc.']
        :phone = +1 (408) 996-1010
        .created = 2024/04/19 14:27:51.696
```

In the example above, the subquery is used to lift the specified industry nodes (`ou:industry`) and assign both nodes' guid values to the `ou:org:industries` property for Apple's organization node.

Note: The `ou:org:industries` property is an **array** (a list or set of typed forms), so the query above uses array-specific syntax. See the *array* section of the *Storm Reference - Type-Specific Storm Behavior* guide for detail on the syntax used to add or modify array properties.

Usage Notes:

- When using a subquery to assign a property value, Storm will throw an error if the subquery fails to lift any nodes.
- When using a subquery to assign a value to a property that takes only a single value, Storm will throw an error if the subquery returns more than one node.
- When using a subquery to assign a property value, the subquery cannot iterate more than 128 times or Storm will throw an error. For example, attempting to assign "all the industries" to a single organization (`ou:org=<guid> [ :industries+={ ou:industry } ]`) will error if there are more than 128 `ou:industry` nodes.

Delete Properties

Operation to delete (fully remove) one or more properties from the specified node(s).

Note: In Synapse 2.x, a property must be deleted from the *Layer* where the property exists. Generally speaking, this means you must be in the *View* where the relevant layer is the topmost (writeable) layer in order to delete the property.

Syntax:

<*query*> **[-:** <*prop*> ... **]**

Examples:

Delete the :loc property from an inet:ipv4 node:

```
<inet:ipv4> [ -:loc ]
```

Delete multiple properties from a media:news node:

```
<media:news> [ -:author -:summary ]
```

Usage Notes:

- Deleting a property fully removes the property from the node; it does not set the property to a null value.

Delete Nodes

Nodes can be deleted from a Cortex using the Storm *delnode* command.

Add Light Edges

Operation that links the specified node(s) to another node or set of nodes (as specified by a Storm expression) using a lightweight edge (light edge).

See *Lightweight (Light) Edge* for details on light edges.

Syntax:

<*query*> **[+(** <*verb*> **)> {** <*storm*> **}]**

<*query*> **[<(** <*verb*> **)+ {** <*storm*> **}]**

Note: The query syntax used to create light edges will yield the nodes that are **inbound to the edit brackets** (that is, the nodes represented by <*query*>).

The nodes specified by the Storm expression (`{ <storm> }`) must already exist in the Cortex or must be created as part of the Storm expression (i.e., using edit brackets) in order for the light edges to be created.

Examples:

Link the specified FQDN and IPv4 to the media:news node referenced by the Storm expression using a "refs" light edge:

```
inet:fqdn=woot.com inet:ipv4=1.2.3.4 [ <(refs)+ {␣
↪media:news=a3759709982377809f28fc0555a38193 } ]
```

Link the specified media:news node to the set of indicators tagged APT1 (#rep.mandiant.apt1) using a "refs" (references) light edge:

```
media:news=a3759709982377809f28fc0555a38193 [ +(refs)> { +#rep.mandiant.apt1 } ]
```

Link the inet:whois:iprec netblock registration (whois) record to any IP address within the specified netblock range (as referenced by the Storm expression) that already exists in Synapse using an "ipwhois" light edge:

```
inet:whois:iprec:name=OVH-CUST-3399212 [ +(ipwhois)> { inet:ipv4=198.50.240.220-198.50.
↪240.223 } ]
```

Link the inet:whois:iprec netblock registration (whois) record to every IP in the specified netblock range (as referenced by the Storm expression) using an "ipwhois" light edge, creating the IPs if they do not exist:

```
inet:whois:iprec:name=OVH-CUST-3399212 [ +(ipwhois)> { [ inet:ipv4=198.50.240.220-198.50.
↪240.223 ] } ]
```

Usage Notes:

- The plus sign (+) used with the light edge expression within the edit brackets is used to create the light edge(s).
- Light edges can be created in either "direction" (e.g., with the directional arrow pointing either right (`+(<verb>)>`) or left (`<(<verb>)+`) - whichever syntax is easier.
- Synapse does not include any pre-existing light edges / light edge verbs. Users can create and define light edges to meet their needs.
- Synapse's data model documentation for *Forms* lists various light edges that can be used with each form, based on The Vertex Project's recommendations. Use of these specific edges is not enforced by Synapse, although these edges may be created by various Vertex-provided Power-Ups.
- Light edge verbs are created "on the fly" when they are first used to link nodes; they do not need to be created or defined in advance before they can be used.
- A light edge's verb typically has a logical direction (a report "references" a set of indicators that it contains, but the indicators do not "reference" the report). It is up to the user to create the light edges in the correct direction and use forms that are sensical for the light edge verb. That is, there is nothing in the Storm syntax itself to prevent users linking arbitrary nodes in arbitrary directions using arbitrary light edges.
- The Storm *edges*, *lift*, and *model* commands can be used to work with light edges in Synapse.

Delete Light Edges

Operation that deletes the light edge linking the specified node(s) to the set of nodes specified by a given Storm expression.

See *Lightweight (Light) Edge* for details on light edges.

Note: In Synapse 2.x, a light edge must be deleted from the *Layer* where the edge exists. Generally speaking, this means you must be in the *View* where the relevant layer is the topmost (writeable) layer in order to delete the edge.

Syntax:

<query> **[-(** *<verb>* **)> {** *<storm>* **}]**

<query> **[<(** *<verb>* **)- {** *<storm>* **}]**

Caution: The minus sign (-) used with a light edge **outside** any edit brackets simply instructs Storm to traverse ("walk") the specified light edge (see *Traversal Operations*). The minus sign used with a light edge **inside** edit brackets instructs Storm to **delete** the specified edges.

Examples:

Delete the "refs" light edge linking the MD5 hash of the empty file to the specified media:news node:

```
hash:md5=d41d8cd98f00b204e9800998ecf8427e [ <(refs)- {␣
↪media:news=a3759709982377809f28fc0555a38193 } ]
```

Delete the "ipwhois" light edge linking IP 1.2.3.4 to the specified netblock registration (whois) record:

```
inet:whois:iprec:name=OVH-CUST-3399212 [ -(ipwhois)> { inet:ipv4=1.2.3.4 } ]
```

Usage Notes:

- The minus sign (-) used with the light edge expression within the edit brackets is used to delete the light edge(s).
- Light edges can be deleted in either "direction" (e.g., with the directional arrow pointing either right (`-(<verb>)>`) or left (`<(<verb>)-`) - whichever syntax is easier.

Add Tags

Operation to add one or more tags to the specified node(s).

Tip: You can optionally use the *"Try" Operator* (`+?#`) when adding tags.

Syntax:

<query> **[+# | +?#** *<tag>* … **]**

Examples:

Add a single tag:

```
<inet:ipv4> [ +#cno.infra.anon.tor.exit ]
```

Add multiple tags:

```
<inet:fqdn> [ +#rep.mandiant.apt1 +#cno.infra.dns.sink.holed ]
```

Add Tag Timestamps or Tag Properties

Synapse supports the use of *Tag Timestamps* and *Tag Properties* to provide additional context to tags where appropriate.

Tip: You can optionally use the *"Try" Operator* when setting or modifying tag timestamps or tag properties.

- When using the try operator with tag timestamps, the operator is used with the tag name (`+?#<tag>=<time>` or `+?#<tag>=(<min_time>,<max_time>)`).
- When using the try operator with a tag property, the operator is used with the tag property value (`+#<tag>:<tagprop>?=<pval>`).

Note that the tag and tag timestamp(s) or the tag and tag property are evaluated as a whole; if any part of the tag expression is invalid, the full edit operation will fail. For example, when attempting to add a tag with timestamps where the tag is valid but the timestamp values are not, neither the tag nor the timestamps will be applied.

Syntax:

Add tag timestamps:

<query> **[+# | +?#** *<tag>* **=** *<time>* **| (** *<min_time>* **,** *<max_time>* **)** ... **]**

Add tag property:

<query> **[+#** *<tag>* **:** *<tagprop>* **= | ?=** *<pval>* ... **]**

Examples:

Add tag with single timestamp:

```
<inet:fqdn> [ +#cno.infra.dns.sink.holed=2018/11/27 ]
```

Add tag with a time interval (min / max):

```
<inet:fqdn> [ +#cno.infra.dns.sink.holed=(2014/11/06, 2016/11/06) ]
```

Tip: Tag timestamps are intervals (ival types). See the *ival* section of the *Storm Reference - Type-Specific Storm Behavior* for details on interval behavior and working with intervals.

Add tag with custom tag property and value:

```
<inet:fqdn> [ +#rep.symantec:risk = 87 ]
```

Tip: Tag properties must be defined and added to the data model before they can be used. See *Tag Properties* for additional information.

Usage Notes:

- *Tag Timestamps* and *Tag Properties* are applied only to the tags to which they are explicitly added. For example, adding a timestamp to the tag `#foo.bar.baz` does **not** add the timestamp to tags `#foo.bar` and `#foo`.

Modify Tags

Tags are "binary" in that they are either applied to a node or they are not. Tag names cannot be changed once set. To "change" the tag applied to a node, you must add the new tag and delete the old one.

Tip: The Storm *movetag* command can be used to modify tags in bulk - that is, migrate an entire set of tags (i.e., effectively "rename" the tags by creating and applying new tags and removing the old ones) or move a tag to a different tag tree.

Modify Tag Timestamps or Tag Properties

Tag timestamps or tag properties can be modified using the same syntax used to add the timestamp or property.

Tip: Tag timestamps are intervals (ival types). See the *ival* section of the *Storm Reference - Type-Specific Storm Behavior* for details on interval behavior when modifying interval values.

Remove Tags

Operation to delete one or more tags from the specified node(s).

Removing a tag from a node differs from deleting the node representing a tag (a `syn:tag` node), which can be done using the Storm *delnode* command.

Note: In Synapse 2.x, a tag must be deleted from the *Layer* where the tag exists. Generally speaking, this means you must be in the *View* where the relevant layer is the topmost (writeable) layer in order to delete the tag.

Syntax:

<*query*> **[-#** <*tag*> … **]**

Examples:

Remove a leaf tag (i.e., the final or rightmost element of the tag):

```
<inet:ipv4> [ -#cno.infra.anon.tor.exit ]
```

Remove a full tag (i.e., the entire tag):

```
<inet:ipv4> [ -#cno ]
```

Usage Notes:

- Deleting a leaf tag deletes **only** the leaf tag from the node. For example, `[ -#foo.bar.baz ]` will delete the tag `#foo.bar.baz` but leave the tags `#foo.bar` and `#foo` on the node.
- Deleting a non-leaf tag deletes that tag and **all tags below it in the tag hierarchy** from the node. For example, `[ -#foo ]` used on a node with tags `#foo.bar.baz` and `#foo.hurr.derp` will remove **all** of the following tags:
  - `#foo.bar.baz`
  - `#foo.hurr.derp`
  - `#foo.bar`
  - `#foo.hurr`
  - `#foo`

Tip: The Storm *tag.prune* command can be used to recursively remove tags (i.e., from a leaf tag up through parent tags that do not have other children).

Remove Tag Timestamps

To remove a tag timestamp from a tag, you must remove the tag element that contains the timestamp. The tag element can be re-added without the timestamp if needed.

Remove Tag Properties

Removing a tag property deletes the property and any property value. The tag element to which the property was appended will remain.

Syntax:

Remove a tag property:

<query> **[-#** *<tag>* **:** *<tagprop>* … **]**

Example:

Remove the custom tag property ':risk' from a tag:

```
<inet:fqdn> [ -#rep.symantec:risk ]
```

Combining Data Modification Operations

Storm allows you to perform multiple edits within a single edit operation (set of edit brackets).

Simple Examples

Create a node and add secondary properties:

```
[ inet:ipv4=94.75.194.194 :loc=nl :asn=60781 ]
```

Create a node and add a tag:

```
[ inet:fqdn=blackcake.net +#rep.mandiant.apt1 ]
```

Edit Brackets and Edit Parentheses Examples

Edit parentheses can be used within edit brackets to isolate edit operations (e.g., so a particular edit does not apply to all inbound nodes).

The following examples illustrate the differences in Storm behavior when using *Edit Brackets* alone vs. with *Edit Parentheses*.

When performing simple edit operations (i.e., Storm queries that add / modify a single node, or apply a tag to the nodes retrieved by a Storm lift operation) users can generally use edit brackets alone without delimiting edit operations within additional edit parentheses (edit parens).

Edit parens may be necessary when creating and modifying multiple nodes in a single query, or performing edits within a longer or more complex Storm query. In these cases, understanding the difference between edit brackets' "operate on everything inbound" vs. edit parens' "limit modifications to the specified nodes" is critical to avoid unintended data modifications.

Example 1:

Consider the following Storm query that uses only edit brackets:

```
inet:fqdn#rep.mandiant.apt1 [ inet:fqdn=somedomain.com +#rep.eset.sednit ]
```

The query will:

- Lift all domains that Mandiant associates with APT1 (i.e., tagged `#rep.mandiant.apt1`).
- Create the new domain `somedomain.com` (if it does not already exist) or lift it (if it does).
- Apply the tag `#rep.eset.sednit` to the domain `somedomain.com` **and** to all of the domains tagged `#rep.mandiant.apt1` (because those FQDNs are inbound to the edit operation / edit brackets).

We can see the effects in the output of our example query:

```
storm> inet:fqdn#rep.mandiant.apt1 [ inet:fqdn=somedomain.com +#rep.eset.sednit ]
inet:fqdn=newsonet.net
        :domain = net
        :host = newsonet
        :issuffix = false
        :iszone = true
        :zone = newsonet.net
        .created = 2024/04/19 14:27:51.887
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.eset.sednit
        #rep.mandiant.apt1
inet:fqdn=staycools.net
        :domain = net
        :host = staycools
        :issuffix = false
        :iszone = true
        :zone = staycools.net
        .created = 2024/04/19 14:27:51.896
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.eset.sednit
        #rep.mandiant.apt1
inet:fqdn=blackcake.net
        :domain = net
        :host = blackcake
        :issuffix = false
        :iszone = true
        :zone = blackcake.net
        .created = 2024/04/19 14:27:52.104
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.eset.sednit
        #rep.mandiant.apt1
inet:fqdn=purpledaily.com
        :domain = com
        :host = purpledaily
        :issuffix = false
        :iszone = true
        :zone = purpledaily.com
        .created = 2024/04/19 14:27:51.909
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.eset.sednit
        #rep.mandiant.apt1
```

(continues on next page)

(continued from previous page)

```
inet:fqdn=hugesoft.org
        :domain = org
        :host = hugesoft
        :issuffix = false
        :iszone = true
        :zone = hugesoft.org
        .created = 2024/04/19 14:27:51.902
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.eset.sednit
        #rep.mandiant.apt1
inet:fqdn=somedomain.com
        :domain = com
        :host = somedomain
        :issuffix = false
        :iszone = true
        :zone = somedomain.com
        .created = 2024/04/19 14:27:52.384
        #rep.eset.sednit
```

Consider the same query using edit parens inside the brackets:

```
inet:fqdn#rep.mandiant.apt1 [ ( inet:fqdn=somedomain.com +#rep.eset.sednit ) ]
```

Because we used the edit parens, the query will:

- Lift all domains that Mandiant associates with APT1 (i.e., tagged `#rep.mandiant.apt1`).
- Create the new domain `somedomain.com` (if it does not already exist) or lift it (if it does).
- Apply the tag `rep.eset.sednit` **only** to the domain `somedomain.com`.

We can see the difference in the output of the example query:

```
storm> inet:fqdn#rep.mandiant.apt1 [ ( inet:fqdn=somedomain.com +#rep.eset.sednit ) ]
inet:fqdn=newsonet.net
        :domain = net
        :host = newsonet
        :issuffix = false
        :iszone = true
        :zone = newsonet.net
        .created = 2024/04/19 14:27:51.887
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.mandiant.apt1
inet:fqdn=staycools.net
        :domain = net
        :host = staycools
        :issuffix = false
        :iszone = true
        :zone = staycools.net
        .created = 2024/04/19 14:27:51.896
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.mandiant.apt1
inet:fqdn=blackcake.net
        :domain = net
```

(continues on next page)

(continued from previous page)

```
        :host = blackcake
        :issuffix = false
        :iszone = true
        :zone = blackcake.net
        .created = 2024/04/19 14:27:52.104
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.mandiant.apt1
inet:fqdn=purpledaily.com
        :domain = com
        :host = purpledaily
        :issuffix = false
        :iszone = true
        :zone = purpledaily.com
        .created = 2024/04/19 14:27:51.909
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.mandiant.apt1
inet:fqdn=hugesoft.org
        :domain = org
        :host = hugesoft
        :issuffix = false
        :iszone = true
        :zone = hugesoft.org
        .created = 2024/04/19 14:27:51.902
        #cno.infra.dns.sink.holed = (2014/11/06 00:00:00.000, 2018/11/27 00:00:00.001)
        #rep.mandiant.apt1
inet:fqdn=somedomain.com
        :domain = com
        :host = somedomain
        :issuffix = false
        :iszone = true
        :zone = somedomain.com
        .created = 2024/04/19 14:27:52.384
        #rep.eset.sednit
```

Example 2:

Consider the following Storm query that uses only edit brackets:

```
[inet:ipv4=1.2.3.4 :asn=1111 inet:ipv4=5.6.7.8 :asn=2222]
```

The query will:

- Create (or lift) the IP address `1.2.3.4`.
- Set the IP's `:asn` property to `1111`.
- Create (or lift) the IP address `5.6.7.8`.
- Set the `:asn` property for **both** IP addresses to `2222`.

We can see the effects in the output of our example query:

```
storm> [inet:ipv4=1.2.3.4 :asn=1111 inet:ipv4=5.6.7.8 :asn=2222]
inet:ipv4=1.2.3.4
        :asn = 2222
```

(continues on next page)

(continued from previous page)

```
        :type = unicast
        .created = 2024/04/19 14:27:52.505
inet:ipv4=5.6.7.8
        :asn = 2222
        :type = unicast
        .created = 2024/04/19 14:27:52.511
```

Consider the same query using edit parens inside the brackets:

```
[ (inet:ipv4=1.2.3.4 :asn=1111) (inet:ipv4=5.6.7.8 :asn=2222) ]
```

Because the brackets separate the two sets of modifications, IP `1.2.3.4` has its `:asn` property set to `1111` while IP `5.6.7.8` has its `:asn` property set to `2222`:

```
storm> [ (inet:ipv4=1.2.3.4 :asn=1111) (inet:ipv4=5.6.7.8 :asn=2222) ]
inet:ipv4=1.2.3.4
        :asn = 1111
        :type = unicast
        .created = 2024/04/19 14:27:52.505
inet:ipv4=5.6.7.8
        :asn = 2222
        :type = unicast
        .created = 2024/04/19 14:27:52.511
```

3.6.7 Storm Reference - Subqueries

This section discusses the following topics:

- *Subquery*
- *Subquery Filter*
- *Using Subqueries to Reference Nodes*

Subquery

A **subquery** is a Storm query that is executed inside of another Storm query. Curly braces (`{ }`) are used to enclose the embedded query.

Note: Curly braces are a Storm syntax element that simply indicates "a Storm query is enclosed here". They can be used to denote a subquery, but have other uses as well.

Recall from *Storm Operating Concepts* that a Storm query can consist of multiple elements (lift, filter, pivot, pipe to command, etc.). This sequence of Storm operations acts as a "pipeline" through which the nodes in the query pass. Regardless of the number of nodes you start with (i.e., the number of nodes in your initial lift), each node is processed individually by each element in the query, from left to right.

The elements in the query can be thought of as "gates". The nodes "inbound" to each gate are processed by that gate in some way. For example, if the "gate" is a filter operation, some nodes may be allowed to pass, while others are dropped ("consumed"), based on the filter. If the gate is a pivot, the inbound node is dropped while the node that is the "target" of the pivot is picked up and added to the pipeline.

Note that in a standard Storm query (as described above) the set of nodes at any given point in the query is constantly changing - the "working set" of nodes is transformed by the various operations. The nodes that "go in" to a particular operation in the query are generally not the same ones that "come out". Note also that as described, this process is **linear** (hence "pipeline").

A **subquery** is another element that can be used as part of a longer Storm query, only in this case the "element" is itself an entire Storm query (as opposed to a filter, pivot, or Storm command).

One advantage of a subquery is that the actions that occur inside the subquery do not affect the "main" Storm execution pipeline - the nodes that "go in" to a subquery are the same nodes that "come out", regardless of what operations occur within the subquery itself. (In terms of the *Storm Operating Concepts*, subqueries do not **consume** nodes by default.) In this way, a subquery can allow you to "branch off" the main Storm execution pipeline, "do a thing" off to the side, and then return to the main execution pipeline as though nothing happened; you resume at the point you left off, with the same set of nodes in the pipeline as when you left.

If you want the nodes that result from the subquery operations to be returned, the `yield` option can be used to do so. Note that yielding the subquery nodes is **in addition to** the set of nodes that passed in to the subquery (not "instead of" the inbound nodes). If you **only** want the nodes resulting from the subquery, you probably don't need a subquery and can just use a standard Storm query instead.

Note: Any **actions** performed inside of a subquery will persist. For example, any modifications made to nodes inside a subquery (setting or modifying properties, applying tags, even creating new nodes) will remain; those changes will be present in the Cortex.

In addition, when setting or updating a *Variable* inside a subquery, the variable can pass back "out" of the subquery and be available to the main Storm query.

What remains unchanged is that the set of nodes inbound to the subquery will be the same set of nodes available (inbound) to the next element in the main Storm query - whatever happens inside the subquery does not affect the set of nodes in the **pipeline** (barring the use of `yield` of course).

This ability to "do a thing off to the side" inside of a Storm query pipeline can add efficiencies to certain queries, allowing you to perform some action inline that would otherwise require a second, separate query to perform. While subqueries have their uses in "standard" Storm, they are particularly useful for more advanced Storm use cases involving variables and control flow.

Note: A subquery is typically used to perform some action related to the Storm query in which it is embedded. But there is no requirement for this to be the case. The subquery can contain any valid Storm, so you could (for example) write a subquery that lifts ten arbitrary email addresses (`{ inet:email | limit 10 }`) in the middle of a longer query. There's not much point to this, but Storm will dutifully lift the nodes, discard them (unless the `yield` option is used), and continue on.

Finally, one important characteristic of a subquery is that **it requires inbound nodes in order to execute.** That is, the subquery is meant to be an element in a larger Storm pipeline, not a stand-alone query, and not the first element in a longer query. Even though the subquery does not affect the inbound nodes (that is, the nodes "pass through" the subquery and are still available as inbound nodes to the next query element), nodes must still be "fired into" the subquery for the subquery action(s) to take place.

For example, the following query will return zero nodes, even though the `yield` directive is present. Because no nodes are "inbound" to cause the subquery to execute, the embedded Storm is never run:

```
storm> yield { inet:email | limit 10 }
```

Syntax:

<*query*> [**yield**] **{** <*query*> **}** [<*query*>]

<*query*> [**yield**] **{** <*query*> [**{** <*query*> **}**] **}** [<*query*>]

Examples:

- Pivot from a set of DNS A records to their associated IPs and then to additional DNS A records associated with those IPs. Use a subquery to check whether any of the IPs are RFC1918 addresses (i.e., have `:type=private`) and if so, tag the IP as non-routable.

```
<inet:dns:a> -> inet:ipv4 { +:type=private [ +#nonroutable ] } -> inet:dns:a
```

- Pivot from a set of IP addresses to any servers associated with those IPs. Use a subquery to check whether the IP has a location (`:loc`) property, and if not, call a third-party geolocation service to attempt to identify a location and set the property. (**Note:** Synapse does not include a geolocation service in its public distribution; this example assumes such a service has been implemented and is called using an extended Storm command named `ipgeoloc`.)

```
<inet:ipv4> { -:loc | ipgeoloc } -> inet:server
```

- Pivot from a set of FQDNs to any files (binaries) that query those FQDNs. Use a subquery with the `yield` option to return the file nodes as well as the original FQDNs.

```
<inet:fqdn> yield { -> inet:dns:request:query:name +:exe -> file:bytes }
```

Note: The "pivot and join" operator (`-+>`) allows you to combine a set of inbound nodes with the set of nodes reached by the pivot into a single result set. However, the operator only allows you to join sets of nodes that are "one degree" (one pivot) apart. The subquery syntax above effectively allows you to join two sets of nodes that are more than one pivot apart.

Usage Notes:

- Subqueries can be nested; you can place subqueries inside of subqueries.
- When the `yield` option is used, Storm will return the nodes from the subquery first, followed by the nodes from the original working set.

Subquery Filter

A **subquery filter** is a filter where the filter itself is a Storm expression.

Standard Storm filter operations are designed to operate on the nodes in the current working set, that is, the nodes actively passing through the Storm pipeline. Regardless of whether the filter uses a *Standard Comparison Operator* or *Extended Comparison Operator*, the filter evaluates some aspect of the node itself such as its primary or secondary property(ies), or whether or not the node has a particular tag.

A subquery filter allows you to use a subquery to filter the current set of nodes based on their relationship to other nodes, or on the properties or tags of "nearby" nodes. The subquery content is still evaluated "off to the side"; any pivots, filters, or other operations performed inside the subquery are still "contained within" the subquery. But the nodes passing through the main Storm pipeline are **evaluated against** the contents of the subquery, and are then filtered - passed or dropped - based on that evaluation.

For additional detail on subquery filters and examples of their use, refer to the *Subquery Filters* section of the *Storm Reference - Filtering* guide.

Using Subqueries to Reference Nodes

A subquery can be used as an alternative (and potentially simpler) way to reference a node in Synapse.

A common use case is to use a subquery as a simpler way to refer to a *GUID Form* or *Composite Form* without needing to enter (type, copy and paste) the form's primary property.

Examples:

Lift all of the contacts (ps:contact nodes) for The Vertex Project:

```
ps:contact:org = { ou:org:name = vertex }
```

Set the :id:number property of a contact (ps:contact node) to the ID number whose value is 444-44-4444:

```
ps:contact:name='ron the cat' [ :id:number = { ou:id:number:value = 444-44-4444 } ]
```

Tip: When a subquery is used to specify a property value (as in the queries above), Synapse will generate an error if the subquery expression returns more than node. Specifically, the `:org` property of a `ps:contact` node is a single `ou:org` guid. If `ou:org:name = vertex` returns more than one org node, Synapse returns a `BadTypeValu` error.

See the *Add or Modify Properties Using Subqueries* section of the *Storm Reference - Data Modification* document for additional examples of setting properties using subqueries.

3.6.8 Storm Reference - Model Introspection

This section provides a brief overview / tutorial of some basic Storm queries to allow introspection / navigation of Synapse's:

- *Data Model*
- *Analytical Model*

The sample queries below are meant to help users new to Synapse and Storm get started examining forms and tags within a Cortex. The queries all use standard Storm syntax and operations (such as pivots). For more detail on using Storm, see *Storm Reference - Introduction* and related Storm topics.

Data Model

Analysts working with the data in the Synapse hypergraph will quickly become familiar with the forms they work with most often. However, as the model expands - or when first learning Synapse - it is helpful to be able to easily reference forms that may be less familiar, as well as how different forms relate to each other.

While the data model can be referenced within the Synapse source code or via the auto-generated *Synapse Data Model* documentation, it can be inconvenient to stop in the middle of an analytical workflow to search for the correct documentation. It is even more challenging to stop and browse through extensive documentation when you're not sure what you're looking for (or whether an appropriate form exists for your needs).

For these reasons Synapse supports **data model introspection** within the Synapse hypergraph itself - that is, the Synapse data model is itself data stored within the Cortex. Introspection allows users to obtain the model definition for a given Cortex at run-time. The model definition contains a list of all native and custom types, forms, and properties supported by the current Cortex.

These model elements are generated as nodes in the Cortex from the current Synapse data model when a Cortex is initialized or when a new module is loaded. As nodes, they can be lifted, filtered, and pivoted across just like other nodes. However, the model-specific nodes do not persist permanently in storage and they cannot be modified (edited) or tagged. Because they are generated at run-time they are known as run-time nodes or **runt nodes.**

The following runt node forms are used to represent the Synapse data model for types, forms, and properties, respectively.

- syn:type
- syn:form
- syn:prop

As nodes within the Cortex, these forms can be lifted, filtered, and pivoted across using the Storm query language, just like any other nodes (with the exception of editing or tagging). Refer to the various Storm documents for details on Storm syntax. A few simple example queries are provided below to illustrate some common operations for model introspection.

Example Queries

- Display all current types / forms / properties:

```
storm> syn:type | limit 2
syn:type=int
        :ctor = synapse.lib.types.Int
        :doc = The base 64 bit signed integer type.
        :opts = {'size': 8, 'signed': True, 'enums:strict': True, 'fmt': '%d', 'min':␣
↪None, 'max': None, 'ismin': False, 'ismax': False}
syn:type=float
        :ctor = synapse.lib.types.Float
        :doc = The base floating point type.
        :opts = {'fmt': '%f', 'min': None, 'minisvalid': True, 'max': None, 'maxisvalid
↪': True}
```

```
storm> syn:form | limit 2
syn:form=inet:dns:a
        :doc = The result of a DNS A record lookup.
        :runt = false
        :type = inet:dns:a
syn:form=inet:dns:aaaa
        :doc = The result of a DNS AAAA record lookup.
        :runt = false
        :type = inet:dns:aaaa
```

```
storm> syn:prop | limit 2
syn:prop=.seen
        :base = .seen
        :doc = The time interval for first/last observation of the node.
        :extmodel = false
        :relname = .seen
        :ro = false
        :type = ival
        :univ = true
syn:prop=.created
        :base = .created
```

(continues on next page)

(continued from previous page)

```
        :doc = The time the node was created in the cortex.
        :extmodel = false
        :relname = .created
        :ro = true
        :type = time
        :univ = true
```

- Display all types that are sub-types of 'string':

```
storm> syn:type:subof = str | limit 2
syn:type=ou:sic
        :ctor = synapse.lib.types.Str
        :doc = The four digit Standard Industrial Classification Code.
        :opts = {'enums': None, 'regex': '^[0-9]{4}$', 'lower': False, 'strip': False,
↪'replace': (), 'onespace': False, 'globsuffix': False}
        :subof = str
syn:type=ou:naics
        :ctor = synapse.lib.types.Str
        :doc = North American Industry Classification System codes and prefixes.
        :opts = {'enums': None, 'regex': '^[1-9][0-9]{1,5}?$', 'lower': False, 'strip':␣
↪True, 'replace': (), 'onespace': False, 'globsuffix': False}
        :subof = str
```

- Display a specific type:

```
storm> syn:type = inet:fqdn
syn:type=inet:fqdn
        :ctor = synapse.models.inet.Fqdn
        :doc = A Fully Qualified Domain Name (FQDN).
```

- Display a specific form:

```
storm> syn:form = inet:fqdn
syn:form=inet:fqdn
        :doc = A Fully Qualified Domain Name (FQDN).
        :runt = false
        :type = inet:fqdn
```

- Display a specific property of a specific form:

```
storm> syn:prop = inet:ipv4:loc
syn:prop=inet:ipv4:loc
        :base = loc
        :doc = The geo-political location string for the IPv4.
        :extmodel = false
        :form = inet:ipv4
        :relname = loc
        :ro = false
        :type = loc
        :univ = false
```

- Display a specific form and all its secondary properties (including universal properties):

```
storm> syn:prop:form = inet:fqdn | limit 2
syn:prop=inet:fqdn
        :doc = A Fully Qualified Domain Name (FQDN).
        :extmodel = false
        :form = inet:fqdn
        :type = inet:fqdn
        :univ = false
syn:prop=inet:fqdn.seen
        :base = .seen
        :doc = The time interval for first/last observation of the node.
        :extmodel = false
        :form = inet:fqdn
        :relname = .seen
        :ro = false
        :type = ival
        :univ = false
```

- Display all properties whose type is `inet:fqdn`:

```
storm> syn:prop:type = inet:fqdn | limit 2
syn:prop=inet:dns:a:fqdn
        :base = fqdn
        :doc = The domain queried for its DNS A record.
        :extmodel = false
        :form = inet:dns:a
        :relname = fqdn
        :ro = true
        :type = inet:fqdn
        :univ = false
syn:prop=inet:dns:aaaa:fqdn
        :base = fqdn
        :doc = The domain queried for its DNS AAAA record.
        :extmodel = false
        :form = inet:dns:aaaa
        :relname = fqdn
        :ro = true
        :type = inet:fqdn
        :univ = false
```

- Display all forms **referenced by** a specific form (i.e., the specified form contains secondary properties that are themselves forms):

```
storm> syn:prop:form = inet:whois:rec :type -> syn:form
syn:form=inet:whois:rec
        :doc = A domain whois record.
        :runt = false
        :type = inet:whois:rec
syn:form=inet:fqdn
        :doc = A Fully Qualified Domain Name (FQDN).
        :runt = false
        :type = inet:fqdn
syn:form=inet:whois:rar
        :doc = A domain registrar.
```

(continues on next page)

(continued from previous page)

```
        :runt = false
        :type = inet:whois:rar
syn:form=inet:whois:reg
        :doc = A domain registrant.
        :runt = false
        :type = inet:whois:reg
```

- Display all forms that **reference** a specific form (i.e., the specified form is a secondary property of another form):

```
storm> syn:form = inet:whois:rec -> syn:prop:type :form -> syn:form
syn:form=inet:whois:contact
        :doc = An individual contact from a domain whois record.
        :runt = false
        :type = inet:whois:contact
syn:form=inet:whois:rec
        :doc = A domain whois record.
        :runt = false
        :type = inet:whois:rec
syn:form=inet:whois:recns
        :doc = A nameserver associated with a domain whois record.
        :runt = false
        :type = inet:whois:recns
```

Analytical Model

As the number of tags used in the hypergraph increases, analysts must be able to readily identify tags, tag hierarchies, and the precise meaning of individual tags so they can be applied and interpreted correctly.

Unlike the runt nodes used for the Synapse data model, the `syn:tag` nodes that represent tags are regular objects in the Cortex that can be lifted, filtered, and pivoted across (as well as edited, tagged, and deleted) just like any other nodes. In a sense it is possible to perform **"analytical model introspection"** by examining the nodes representing a Cortex's analytical model (i.e., tags).

Lifting, filtering, and pivoting across `syn:tag` nodes is performed using the standard Storm query syntax; refer to the various Storm documents for details on using Storm. See also the `syn:tag` section of *Storm Reference - Type-Specific Storm Behavior* for additional details on working with `syn:tag` nodes.

A few simple example queries are provided below to illustrate some common operations for working with tags. As Synapse does not include any pre-populated `syn:tag` nodes, these examples assume you have a Cortex where some number of tags have been created.

Example Queries

- Lift a single tag:

```
storm> syn:tag = cno.infra.anon.tor
syn:tag=cno.infra.anon.tor
        :base = tor
        :depth = 3
        :doc = Various types of Tor infrastructure, including: a server representing a␣
↪Tor service or the associated IP address; a host known to be a Tor node / hosting a␣
↪Tor service; contact information associated with an entity responsible for a given Tor␣
```

(continues on next page)

(continued from previous page)

```
↪node.
        :title = Tor Infrastructure
        :up = cno.infra.anon
        .created = 2024/04/19 14:26:53.422
```

- Lift all root tags:

```
storm> syn:tag:depth = 0
syn:tag=cno
        :base = cno
        :depth = 0
        .created = 2024/04/19 14:26:53.409
```

- Lift all tags one level "down" from the specified tag:

```
storm> syn:tag:up = cno.infra.anon
syn:tag=cno.infra.anon.vpn
        :base = vpn
        :depth = 3
        :doc = A server representing an anonymous VPN service, or the associated IP␣
↪address. Alternately, an FQDN explicilty denoting an anonymous VPN that resolves to␣
↪the associated IP.
        :title = Anonymous VPN
        :up = cno.infra.anon
        .created = 2024/04/19 14:26:53.428
syn:tag=cno.infra.anon.tor
        :base = tor
        :depth = 3
        :doc = Various types of Tor infrastructure, including: a server representing a␣
↪Tor service or the associated IP address; a host known to be a Tor node / hosting a␣
↪Tor service; contact information associated with an entity responsible for a given Tor␣
↪node.
        :title = Tor Infrastructure
        :up = cno.infra.anon
        .created = 2024/04/19 14:26:53.422
```

- Lift all tags that start with a given prefix, regardless of depth:

```
storm> syn:tag ^= cno.infra
syn:tag=cno.infra
        :base = infra
        :depth = 1
        :doc = Top-level tag for infrastructre.
        :title = Infrastructure
        :up = cno
        .created = 2024/04/19 14:26:53.409
syn:tag=cno.infra.anon
        :base = anon
        :depth = 2
        :doc = Top-level tag for anonymization services.
        :title = Anonymization services
        :up = cno.infra
        .created = 2024/04/19 14:26:53.416
```

(continues on next page)

(continued from previous page)

```
syn:tag=cno.infra.anon.tor
        :base = tor
        :depth = 3
        :doc = Various types of Tor infrastructure, including: a server representing a␣
↪Tor service or the associated IP address; a host known to be a Tor node / hosting a␣
↪Tor service; contact information associated with an entity responsible for a given Tor␣
↪node.
        :title = Tor Infrastructure
        :up = cno.infra.anon
        .created = 2024/04/19 14:26:53.422
syn:tag=cno.infra.anon.vpn
        :base = vpn
        :depth = 3
        :doc = A server representing an anonymous VPN service, or the associated IP␣
↪address. Alternately, an FQDN explicilty denoting an anonymous VPN that resolves to␣
↪the associated IP.
        :title = Anonymous VPN
        :up = cno.infra.anon
        .created = 2024/04/19 14:26:53.428
```

- Lift all tags that share the same base (rightmost) element:

```
storm> syn:tag:base = sofacy
syn:tag=rep.talos.sofacy
        :base = sofacy
        :depth = 2
        :doc = Indicator or activity talos calls (or associates with) sofacy.
        :title = sofacy(talos)
        :up = rep.talos
        .created = 2024/04/19 14:26:53.540
syn:tag=rep.uscert.sofacy
        :base = sofacy
        :depth = 2
        :doc = Indicator or activity uscert calls (or associates with) sofacy.
        :title = sofacy(uscert)
        :up = rep.uscert
        .created = 2024/04/19 14:26:53.534
```

3.6.9 Storm Reference - Type-Specific Storm Behavior

Some data types (*Type*) within Synapse have additional optimizations. These include optimizations for:

- indexing (how the type is stored for retrieval);
- parsing (how the type can be specified for input);
- insertion (how the type can be used to create or modify nodes);
- operations (how the type can be lifted, filtered, or otherwise compared).

Types that have been optimized in various ways are documented below along with any specialized operations that may be available for those types.

This section is **not** a complete reference of all available types. In addition, this section does **not** address the full range of type enforcement constraints that may restrict the values that can be specified for a given type (such as via a constructor (`ctor`)). For details on available types and type constraints or enforcement, see the online documentation or the Synapse source code.

- *array* (array)
- *file:bytes* (file)
- *guid* (globally unique identifier)
- *inet:fqdn* (FQDN)
- *inet:ipv4* (IPv4)
- *int* (integer)
- *ival* (time interval)
- *loc* (location)
- *str* (string)
- *syn:tag* (tag)
- *time* (date/time)

array

An `array` is a specialized type that consists of either a list or a set of typed values. That is, an array is a type that consists of one or more values that are themselves all of a single, defined type.

Tip: An array that is a **list** can have duplicate entries in the list. An array that is a **set** consists of a unique group of entries.

`Array` types can be used for properties where that property is likely to have multiple values, but it is undesirable to represent those values using multiple *Relationship* nodes. Examples of array secondary properties include `media:news:authors`, `inet:email:message:headers`, and `ps:person:names`. You can view all secondary properties that are `array` types using the following Storm query:

```
syn:prop:type=array
```

Indexing

N/A

Parsing

Because an `array` is a list or set of typed values, `array` elements can be input in any format supported by the type of the elements themselves. For example, if an `array` consists of `inet:ipv4` values, the values can be input in any supported `inet:ipv4` format (e.g., integer, hex, dotted-decimal string, etc.).

Insertion

Because it may contain multiple values, an `array` property must be set using comma-separated values enclosed in parentheses (this is true even if the array contains only a single element; you must still use parentheses, and the single element must still be followed by a trailing comma). Single or double quotes are required in accordance with the standard rules for using *Whitespace and Literals in Storm*.

Example:

Set the `:names` property of an organization (`ou:org`) node to a single value:

```
storm> ou:org:name=vertex [ :names=('The Vertex Project',) ]
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
```

Example:

Set the `:names` property of an organization (`ou:org`) node to contain multiple variations of the organization name:

```
storm> ou:org:name=vertex [ :names=('The Vertex Project', 'The Vertex Project, LLC',␣
↪Vertex) ]
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
```

> **Warning:** Using the equals (=) operator to set an array property value will set or update (overwrite) the **entire** property value. To add or remove individual elements from an array, use the += or -= operators.

Example:

Add a name to the array of names associated with an organization:

```
storm> ou:org:name='Monty Python' [ :names+='The Spanish Inquisition' ]
ou:org=5f9168f349c2ab26a31377aef4fbb3c0
        :name = monty python
        :names = ['monty python', 'the spanish inquisition']
        .created = 2024/04/19 14:27:09.297
```

Remove a name from the array of names associated with an organization:

```
storm> ou:org:name='Monty Python' [ :names-='The Spanish Inquisition' ]
ou:org=5f9168f349c2ab26a31377aef4fbb3c0
        :name = monty python
        :names = ['monty python']
        .created = 2024/04/19 14:27:09.297
```

Tip: The standard "edit try" operator (?=) (see *"Try" Operator* in the *Storm Reference - Data Modification*) can be used to attempt to set a **full** array property value where you are unsure whether the value will succeed. The specialized

?+= or ?-= operators can be used to attempt to add or remove a **single** array value in a similar manner.

Example:

Use the specialized "edit try" operator to attempt to add a single value to the :authors array property of an article (media:news node). (**Note:** a type-inappropriate value (a name) is used below to show the "fail silently" behavior for the "edit try" operator. The :authors property is an array of ps:contact nodes and requires ps:contact guid values.)

```
storm> media:news:org=kaspersky [ :authors?+='john smith' ]
media:news=090d157766c1018ff86d69e34f34e28c
        :org = kaspersky
        :title = new report on really bad threat
        .created = 2024/04/19 14:27:09.364
```

Usage Notes:

- When using the standard "edit try" operator (?=) to attempt to set the **full** value of an array property (vs. adding or removing an element from an array), the **entire** attempt will fail if **any** value in the list of values fails. For example, if you try to set [:identities:emails?=(alice@vertex.link, bob)] on an X509 certificate (crypto:x509:cert), Synapse will fail to set the property altogether because bob is not a valid email address type (even though alice@vertex.link is).
- The "edit try" operator for **removing** individual elements from an array (?-=) is unique to arrays as they are the only type that allows removal of a single element from a property. (Properties with a single value are either set, modified (updated), or the property is deleted altogether.) As with other uses of "edit try", use of the operator allows the operation to silently fail (vs. error and halt) if the operation attempts to remove a value from an array that does not match the array's defined type. For example, attempting to remove an IPv4 from an array of email addresses will halt with a BadTypeValu error if the standard remove operator (-=) is used, but silently fail (do nothing and continue) if the "edit try" version (?-=) is used.

Operations

Lifting and Filtering

Lifting or filtering array properties using the equals (=) operator requires an **exact match** of the full array property value. This makes sense for forms with simple values like inet:ipv4=1.2.3.4, but is often infeasible for arrays because lifting by the **full** array value requires you to know the **exact** values of each of the array elements as well as their **exact** order:

```
storm> ou:org:names=("The Vertex Project", "The Vertex Project, LLC", Vertex)
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
```

For this reason, Storm offers a special "by" syntax for lifting and filtering with array types. The syntax consists of an asterisk (*) preceding a set of square brackets ([ ]), where the square brackets contain a comparison operator and a value that can match one or more elements in the array. This allows users to match one or more elements in the array similarly to how they would match individual property values.

Note: The square brackets used to lift or filter based on values in an array should not be confused with square brackets used to add or modify nodes or properties in *Edit Mode*.

Examples:

Lift the `ou:org` node(s) whose `:names` property contains a name that exactly matches `vertex`:

```
storm> ou:org:names*[=vertex]
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
```

Lift the `ou:org` node(s) whose `:names` property contains a name that includes the string `vertex`:

```
storm> ou:org:names*[~=vertex]
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
ou:org=29b6e7bad25fc3538503ba94bd89365a
        :name = vertex
        :names = ['the vertex project', 'the vertex project, llc', 'vertex']
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.204
```

Lift the x509 certificate nodes that reference the domain `microsoft.com`:

```
storm> crypto:x509:cert:identities:fqdns*[=microsoft.com]
crypto:x509:cert=66e6da1a6f72894280e3ef7f8a08b74d
        :identities:fqdns = ['microsoft.com', 'verisign.com']
        .created = 2024/04/19 14:27:09.479
```

Downselect a set of `ou:org` nodes to include only those with a name that starts with “acme”:

```
storm> ou:org +:names*[^=acme]
ou:org=1c6bda5011ba158abab4d6decd73698b
        :name = acme consulting
        :names = ['acme consulting']
        .created = 2024/04/19 14:27:09.532
ou:org=bbb36f4645f031e103b6931b87f18da5
        :name = acme construction
        :names = ['acme construction']
        .created = 2024/04/19 14:27:09.538
```

See *Lift by (Arrays) (*[ ])* and *Filter by (Arrays) (*[ ])* for additional details.

Pivoting

Synapse and Storm are type-aware and will facilitate pivoting between properties of the same type. This includes pivoting between individual typed properties and array properties consisting of those same types. Type awareness for arrays includes both standard form and property pivots as well as wildcard pivots.

Examples:

Pivot from a set of x509 certificate nodes to the set of domains referenced by the certificates (such as in the `:identities:fqdns` array property):

```
storm> crypto:x509:cert -> inet:fqdn
inet:fqdn=microsoft.com
        :domain = com
        :host = microsoft
        :issuffix = false
        :iszone = true
        :zone = microsoft.com
        .created = 2024/04/19 14:27:09.480
inet:fqdn=verisign.com
        :domain = com
        :host = verisign
        :issuffix = false
        :iszone = true
        :zone = verisign.com
        .created = 2024/04/19 14:27:09.480
```

Pivot from a set of `ou:name` nodes to any nodes that reference those names (this would include `ou:org` nodes where the `ou:name` is present in the `:name` property or as an element in the `:names` array):

```
storm> ou:name^=acme <- *
ou:org=bbb36f4645f031e103b6931b87f18da5
        :name = acme construction
        :names = ['acme construction']
        .created = 2024/04/19 14:27:09.538
ou:org=bbb36f4645f031e103b6931b87f18da5
        :name = acme construction
        :names = ['acme construction']
        .created = 2024/04/19 14:27:09.538
ou:org=1c6bda5011ba158abab4d6decd73698b
        :name = acme consulting
        :names = ['acme consulting']
        .created = 2024/04/19 14:27:09.532
ou:org=1c6bda5011ba158abab4d6decd73698b
        :name = acme consulting
        :names = ['acme consulting']
        .created = 2024/04/19 14:27:09.532
```

file:bytes

`file:bytes` is a special type used to represent any file (i.e., any arbitrary set of bytes). Note that a file can be represented as a node within a Cortex regardless of whether the file itself (the specific set of bytes) is available (i.e., stored in an Axon). This is essential as many other data model elements allow (or depend on) the concept of a file (as opposed to a hash).

The `file:bytes` type is a specialized *guid* type. A file can be uniquely represented by the specific contents of the file itself. As it is impractical to use "all the bytes" as a primary property value, it makes sense to use a shortened representation of those bytes - that is, a hash. MD5 collisions can now be generated with ease, and SHA1 collisions were demonstrated in 2017. For this reason, Synapse uses the SHA256 hash of a file (considered sufficiently immune from collision attacks for the time being) as "unique enough" to act as the primary property of a `file:bytes` node if available. Otherwise, a `guid` is generated and used.

Indexing

N/A

Parsing

`file:bytes` must be input using their complete primary property. It is impractical to manually type a SHA256 hash or 128-bit `guid` value. For this reason `file:bytes` forms are most often specified by referencing the node via a more human-friendly secondary property or by pivoting to the node. Alternately, the `file:bytes` value can be copied and pasted for use in a query.

The primary property of a `file:bytes` node indicates how the node was created (i.e., via the SHA256 hash or via a guid):

- A node created using the SHA256 hash will have a primary property value consisting of `sha256:` prepended to the SHA256 hash:

  `file:bytes=sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`

- A node created using a `guid` will have a primary property value consisting of `guid:` prepended to the `guid` value:

  `file:bytes=guid:22d4ed1b75c9eb5ff8070e0df1e8ed6b`

Note: When specifying a SHA256-based `file:bytes` node, entering the `sha256:` prefix is optional. The following are equivalent representations of the same file:

```
file:bytes=sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

file:bytes=e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Insertion

A `file:bytes` node can be created in one of three ways:

SHA256 Hash

A SHA256 hash can be specified as the node's primary property. The `sha256:` prefix can optionally be specified, but is not required (it will be added automatically on node creation). Storm will recognize the primary property value as a SHA256 hash and also set the `:sha256` secondary property. Any other secondary properties must be set manually.

```
storm> [ file:bytes = 44daad9dbd84c92fa9ec52649b028b4c0f7d285407685778d09bad4b397747d0 ]
file:bytes=sha256:44daad9dbd84c92fa9ec52649b028b4c0f7d285407685778d09bad4b397747d0
        :sha256 = 44daad9dbd84c92fa9ec52649b028b4c0f7d285407685778d09bad4b397747d0
        .created = 2024/04/19 14:27:09.642
```

Because the SHA256 is considered unique (for now) for our purposes, the node is fully deconflictable. If additional secondary properties such as `:size` or other hashes are obtained later, or if the actual file is obtained, the node can be updated with the additional properties based on deconfliction with the SHA256 hash.

GUID Value

The asterisk can be used to generate a `file:bytes` node with an arbitrary guid value:

```
storm> [ file:bytes = * ]
file:bytes=guid:61feba3d343f2a1d6adcef5b1b01d128
        .created = 2024/04/19 14:27:09.669
```

Alternately, a potentially deconflictable `guid` can be generated by specifying a list of one or more values to the `guid` generator (for example, an MD5 and / or SHA1 hash). This will generate a predictable `guid`:

```
storm> [ file:bytes = (63fcc49b2ac6cbd686f4d9704446c673,)␣
↪:md5=63fcc49b2ac6cbd686f4d9704446c673 ]
file:bytes=guid:34f71d05b9e06558b184aac6f4010a12
        :md5 = 63fcc49b2ac6cbd686f4d9704446c673
        .created = 2024/04/19 14:27:09.696
```

Synapse does not recognize any strings passed to the `guid` generator as specific types or properties and will not use values used to generate the `guid` to set any secondary property values; those properties must be explicitly set (e.g., the `:md5` property in the example above).

See the section on type-specific behavior for *guid* types for additional discussion of arbitrary (non-deconflictable) vs. deconflictable `guids`.

Note: "Deconflicting" `file:bytes` nodes based on an MD5 or SHA1 hash alone is potentially risky because both of those hashes are subject to collision attacks. In other words, two files that have the same MD5 hash or the same SHA1 hash are not guaranteed to be the same file based on that single hash alone.

In short, creating `file:bytes` nodes using the MD5 and / or SHA1 hash can allow the creation of "potentially" deconflictable nodes when no other data is available. However, this deconfliction is subject to some limitations, as noted above. In addition, if the actual file (full bytes) or corresponding SHA256 hash is obtained later, it is not possible to "convert" a `guid`-based `file:bytes` node to one whose primary property is based on the SHA256 hash.

Actual Bytes

You can also create a `file:bytes` node by adding the actual file (set of bytes) to Synapse (specifically, to Synapse's Axon storage). Adding the file will create the `file:bytes` node in the Cortex based on the file's SHA256 value. Synapse will also calculate and set additional properties for the `file:bytes` node's size and other hashes (e.g., MD5, SHA1, etc.).

Creating `file:bytes` nodes in this manner is often done programmatically (such as via a Synapse *Power-Up*) that can download or ingest files. Other options include:

- the built-in Synapse *wget* command;
- the **Upload File** menu option available from the Synapse UI (Optic), which allows you to either upload a file from local disk, or download a file from a specified URL; or
- the `pushfile` tool, available from the CLI in the community version of Synapse (see *pushfile*).

Tip: Like other external (to Storm) commands, the pushfile tool is accessible from the Storm CLI (see *storm*) as `!pushfile`.

Similarly, Storm's HTTP library (*$lib.inet.http*) could be leveraged to retrieve a web-based file and use the returned bytes as input (potentially using Storm variables - see *Storm Reference - Advanced - Variables*) to the `guid` generator. A detailed discussion of this method is beyond the scope of this section; see the *Storm Libraries* technical documentation for additional detail.

Operations

For some lift and filter operations, you may optionally specify `file:bytes` nodes using a "sufficiently unique" partial match of the node's primary property. For example, the prefix operator (`^=`) may be used to specify a unique prefix for the `file:bytes` node's SHA256 or guid value:

```
storm> file:bytes^=sha256:021b4ce5
file:bytes=sha256:021b4ce5c4d9eb45ed016fe7d87abe745ea961b712a08ea4c6b1b81d791f1eca
        :md5 = 8934aeed5d213fe29e858eee616a6ec7
        :name = adobeupdater.exe
        :sha1 = a7e576f41f7f100c1d03f478b05c7812c1db48ad
        :sha256 = 021b4ce5c4d9eb45ed016fe7d87abe745ea961b712a08ea4c6b1b81d791f1eca
        :size = 182820
        .created = 2024/04/19 14:27:09.719
```

Usage Notes:

- When using the prefix operator, the `sha256:` or `guid:` prefix string must be included.
- The length of the value that is "sufficiently unique" to select a single `file:bytes` will vary depending on the data in your instance of Synapse. If your selection criteria matches more than one `file:bytes` node, Synapse will return all matches.
- Alternatively, the regular expression operator (~=) may be used to specify a partial string match anywhere in the `file:bytes` node's primary property value (though this is an inefficient way to specify a `file:bytes` node).

guid

Within Synapse, a Globally Unique Identifier (`guid`) as a *Type* explicitly refers to a 128-bit value used as a form's primary property.

The term should not be confused with the definition of GUID used by Microsoft, or with other types of identifiers (node ID, task ID) used within Synapse.

The `guid` type is used as the primary property for forms that cannot be uniquely defined by any set of specific properties. See the background documents on the Synapse data model for additional details on the *Guid Form*.

A guid value may be generated arbitrarily or in a predictable (i.e., repeatable) manner based on a defined set of inputs.

See the section on *file:bytes* types for discussion of `file:bytes` as a specialized instance of a `guid` type.

Indexing

N/A

Parsing

Guids must be input using their complete 128-bit value. It is generally impractical to manually type a guid. Guid forms are most often specified by referencing the node via a more human-friendly secondary property. Alternately, the guid value can be copied and pasted.

Insertion

Guids can be generated **arbitrarily** or as **predictable** values. When choosing a method, you should consider how you will **deconflict** guid-based nodes. See *Guid Best Practices* below for additional discussion.

Arbitrary Values

When creating a new guid node, you can specify the asterisk (`*`) as the primary property value of the new node. This tells Synapse to generate a unique, **arbitrary** guid for the node. For example:

```
storm> [ ou:org=* :alias=vertex :name="The Vertex Project" :url=https://vertex.link/ ]
ou:org=bc46b792b2a69bce898428204a99f25b
        :alias = vertex
        :name = the vertex project
        :url = https://vertex.link/
        .created = 2024/04/19 14:27:09.769
```

The above query creates a new organization node with a unique arbitrary guid for its primary property, and sets the specified secondary properties.

> **Warning:** Because the guid generated by the asterisk is **arbitrary,** running the above query a second time will create a second `ou:org` node with a **new** unique guid (potentially resulting in two nodes representing the same organization within the same Cortex).

The advantage of arbitrary values is that they are simple to generate. This is particularly useful for analysts who need to manually create guid nodes (organizations, contacts, threats) on a regular basis as part of their workflow.

The disadvantage is that arbitrary values are truly arbitrary; there is no easy way to deconflict the nodes.

- Users may inadvertently create duplicate nodes. That is, two users can independently create nodes with different guids to represent the same object. The only way to prevent this is by convention - for example, establishing internal processes where users "check first" before creating certain nodes. Note that while this may limit duplication, it is unlikely to eliminate it entirely.
- Bulk data that is ingested using arbitrary guids cannot be reingested, at least not in the same way. Reingesting the same data will create a second set of nodes for the same data but with different arbitrary guids.

Predictable Values

You can generate a guid value in a predictable manner based on a defined set of inputs. The inputs are specified as a comma-separated list within a set of parentheses. The guid generator uses these values as "seed" data to create a **predictable** guid value; the same set of seed data always generates the same guid.

For example:

```
storm> [ ou:org=('the vertex project',https://vertex.link) :name='the vertex project'␣
↪:url=https://vertex.link ]
ou:org=6f08c79ef95d73102af8b4ebca9c22f9
        :name = the vertex project
        :url = https://vertex.link
        .created = 2024/04/19 14:27:09.797
```

The query above creates a new organization node whose guid is generated using the company name and web site as a set of (presumably) unique inputs that will result in a unique (but predictable / deterministic) guid.

The advantage of predictable guids is that they are re-encounterable and therefore deconflictable: if you ingest data using a predictable guid, the same data can be reingested without creating duplicate nodes. This is helpful in cases where a preliminary data set is loaded into Synapse for analysis, and subsequent changes (improvements to the ingest logic, additions to the Synapse data model) allow you to capture additional detail from the original data set.

The disadvantage is that this method is more complicated for users who need to manually create guid nodes. Expecting a group of users to all remember to specify the same set of inputs in the same order (and without typos) each time they create a guid node is unrealistic.

In addition, predictable guids may not fully address challenges associated with ingesting similar data from different sources. Multiple vendors may provide similar information on the same entity. If you obtain data for the same object (an organization, a person, a certificate) from different sources, you may end up with two different nodes if the "predictable" guids are generated with different seed data from each source.

Guid Best Practices

When selecting a method to create guids, a key consideration is how you will **deconflict** data represented by guid forms. Guid forms are unique in that their primary property has no direct or obvious relationship to the object it represents. The primary property `ou:org=44db774d29f27684add0d892931c6e86` tells me this is an organization node, but provides no clue as to whether the organization is The Vertex Project, the World Bank, or the University of Michigan marching band.

The important information about "what" a guid form represents is stored in the form's secondary properties. So from a deconfliction standpoint, the best way to see if a guid node already exists is to use **secondary property deconfliction:**

- Query for an existing node based on one or more meaningful secondary properties.
- The query will lift (return) the selected node(s), if found; otherwise

- Create a new guid node using an arbitrary guid (`*`).

Example

SSL/TLS certificate data is available from various data sources / APIs; different sites or vendors may provide similar information about the same certificate. Certificate metadata (i.e., information such as fingerprints, validity dates, etc.) is represented as a `crypto:x509:cert` node, which is a guid form. If you obtain data about the same certificate from different data sources, you risk the creation of duplicate nodes.

Instead, when ingesting data about a specific certificate, a user (or process) can first check for a `crypto:x509:cert` node based on a unique property, such as a certificate fingerprint (e.g., `crypto:x509:cert:sha1`, or ideally `crypto:x509:cert:sha256` to avoid hash collisions). If an existing node is found, that node can be selected and updated (or otherwise operated on); otherwise a new node for that certificate can be created using an arbitrary guid (`*`) with the appropriate secondary properties set.

Using secondary property deconfliction for guid nodes has the advantage of deconflicting on meaningful properties (those likely to uniquely identify an object), without relying on knowledge of any specific method used to create predictable guids. (Note that "predictable guids" are often generated using these same secondary properties; so deconflicting on the properties directly is both more straightforward and more transparent.)

Tip: When choosing a secondary property (or properties) to deconflict on, you should select ones that can sufficiently deconflict the form **and** are likely to be present in the data source(s) you may use to obtain information about the form.

Secondary property deconfliction is not guaranteed to eliminate all duplications, but is highly effective in many cases. This method can be used both programmatically (i.e., in any ingest scripts or Power-Ups (*Power-Up*)) and by users who can "spot check" for the existence of a node before manually creating one.

Tip: Synapse implements several Storm commands known as generator ("gen") commands. These commands simplify secondary property deconfliction and node creation for several common guid nodes.

For example, the *gen.ou.org* command takes an organization name as input (e.g, "vertex"), checks for any `ou:org` nodes with that name (i.e., in the `:name` or `:names` properties) and either lifts the existing node, or creates a new one.

See the *gen* section in the *Storm Reference - Storm Commands* for available generator commands (or run `help` from your Synapse CLI).

Arbitrary Guids

For some use cases, the use of arbitrary guids (without secondary property deconfliction) may be reasonable. This approach may be suitable when:

- The data you are ingesting is truly unique (i.e., the same or similar data is not available from another source). For example, log or alert data that is specific to a unique sensor or host.
- You need to perform a one-time ingest of the data (i.e., you do not plan to reingest the same data in the future).

If the data is unique, but you may need to reingest it at some point, secondary property deconfliction or predictable guids are more appropriate.

Predictable Guids

For some use cases, the use of predictable guids (without secondary property deconfliction) may be reasonable. This approach may be suitable when:

- You have a unique set of data (not available from another data source) to ingest and want the option to reingest it in the future without creating duplicate nodes.
- The data is sufficiently unique that nodes representing the data will not already exist in Synapse.
- You cannot use secondary property deconfliction given the nature of the data. In this case, deconfliction based on predictable guids may be the "next best" option.

When using predictable guids, the "seed" data to generate the guid should be unique to both the node being created and the specific data source. For example, your inputs could include:

- A string representing the **data source.**
- The **timestamp** associated with the data, if one exists.
- The values of one or more **secondary properties** for the node you are creating. Be sure to choose properties where:
  - the property / properties will **always** be present for the given data source; and
  - the set of properties is sufficient to create a unique node.

For example, a `media:news` node might be created using:

- A data source string (e.g., `my_data_source`).
- The publication date of the article (e.g., `2022/09/12`)
- The URL where the article was published (e.g., `https://www.example.com/my_article.html`)

Predictable guid values can be generated directly (as part of Storm *Edit Mode* syntax):

```
storm> [ media:news=(my_data_source,2022/09/12,https://www.example.com/my_article.html) ]
media:news=f9515b24f615448ed44601645d547f6a
        .created = 2024/04/19 14:27:09.825
```

Alternately, guid values can be generated and assigned to a variable using the Storm `$lib.guid()` library (see *$lib.guid(*args, valu=$lib.undef)*). The values provided as arguments to `$lib.guid()` can be either specific values or variables:

```
storm> $guid=$lib.guid(my_data_source,2022/09/12,https://www.example.com/my_article.
↪html) [ media:news=$guid ]
media:news=f9515b24f615448ed44601645d547f6a
        .created = 2024/04/19 14:27:09.825
```

```
storm> $source=my_data_source $published=2022/09/12 $url=https://www.example.com/my_
↪article.html $guid=$lib.guid($source,$published,$url) [ media:news=$guid ]
media:news=f9515b24f615448ed44601645d547f6a
        .created = 2024/04/19 14:27:09.825
```

Note that the same guid value is generated in each of the three examples above.

Note: The input to the `guid` generator is interpreted as a **structured list,** specifically, a list of string values (i.e., `(str_0, str_1, str_2...str_n)`). Deconfliction depends on the same list being submitted to the generator in the same order each time.

The guid generator is **not** "model aware" and will not recognize items in the list as having a particular data type or representing a particular property value. That is, Synapse will not set any secondary property values based on data provided to the guid generator. Any property values must be set as part of the node creation process.

A full discussion of writing ingest code (particularly for Storm packages, services, or Power-Ups) is beyond the scope of this User Guide. For more information, see the *Synapse Developer Guide*.

Operations

Because guid values are unwieldy to use on the command line (outside of copy and paste operations), guid nodes may be more easily lifted by a unique secondary property.

Examples:

Lift an org node by its alias:

```
storm> ou:org:alias=choam
ou:org=02da1edc657a88ba3def28846cee6e3b
        :alias = choam
        :name = combine honnete ober advancer mercantiles
        .created = 2024/04/19 14:27:09.928
```

Lift a DNS request node by the name used in the DNS query:

```
storm> inet:dns:request:query:name=pop.seznam.cz
inet:dns:request=6d9b19b5889e097faac23e8a17c19a65
        :query:name = pop.seznam.cz
        :query:name:fqdn = pop.seznam.cz
        :time = 2020/04/30 09:30:33.000
        .created = 2024/04/19 14:27:09.972
```

It is also possible to lift and filter guid nodes using a "sufficiently unique" prefix match of the guid value.

Example:

Lift a ps:contact node by a partial prefix match:

```
storm> ps:contact^=13c9663e
ps:contact=13c9663e5f553014eb50d00bb7c6945a
        :name = seongsu park
        :orgname = kaspersky lab
        .created = 2024/04/19 14:27:10.021
```

The length of the value that is "sufficiently unique" will vary depending on the data in your instance of Synapse. If your selection criteria matches more than one node, Synapse will return all matches.

When **setting** or **updating** a secondary property that is a guid value, you may use a "human friendly" Storm query (specifically a subquery) to reference the node whose primary property (guid value) you wish to set for the secondary property.

Example:

Set the :org property for a ps:contact node to the guid value of the associated ou:org node using a Storm query:

```
storm> ps:contact:name='ron the cat' [ :org={ ou:org:name=vertex } ]
ps:contact=a4050bab7b2ca8822fe1bec7b020dcd3
```

(continues on next page)

(continued from previous page)

```
        :name = ron the cat
        :org = 29b6e7bad25fc3538503ba94bd89365a
        :title = cattribution analyst
        .created = 2024/04/19 14:27:10.065
```

Note: The Storm query used to specify the guid node must return exactly one node. If the query returns more than one node, or does not return any nodes, Synapse will generate an error.

See *Add or Modify Properties Using Subqueries* for additional details.

inet:fqdn

Fully qualified domain names (FQDNs) are structured as a set of string elements separated by the dot (`.`) character. The Domain Name System acts as a "reverse hierarchy" (operating from right to left instead of from left to right) separated along the dot boundaries - i.e., `com` is the hierarchical root for domains such as `google.com` or `microsoft.com`.

Because of this logical structure, Synapse includes certain optimizations for working with `inet:fqdn` types:

- Reverse string indexing on `inet:fqdn` types.
- Default values for the secondary properties `:issuffix` and `:iszone` of a given `inet:fqdn` node based on the values of those properties for the node's parent domain.

Indexing

Synapse performs **reverse string indexing** on `inet:fqdn` types. Domains are indexed in full reverse order - that is, the domain `this.is.my.domain.com` is indexed as `moc.niamod.ym.si.siht` to account for the "reverse hierarchy" implicit in the DNS structure.

Parsing

N/A

Insertion

When `inet:fqdn` nodes are created (or modifications to certain properties are made), Synapse uses some built-in logic to set certain secondary properties related to zones of control (specifically, `:issuffix`, `:iszone`, and `:zone`).

The reverse hierarchy implicit in dotted FQDNs represents elements such as <*host*>.<*domain*>.<*suffix*>, but can also represent implicit or explicit **zones of control.** The term "zone of control" is loosely defined, and is not meant to represent control or authority by any specific organization or entity. Instead, "zone of control" can be thought of as a boundary within an individual FQDN hierarchy where control of a portion of the domain namespace shifts from one entity or owner to another.

A simple example is the `com` top-level domain (managed by Verisign) vs. the domain `microsoft.com` (controlled by Microsoft Corporation). `Com` represents one zone of control where `microsoft.com` represents another.

The `inet:fqdn` form in the Synapse data model uses several secondary properties that relate to zones of control:

- `:issuffix` = primary zone of control

- `:iszone` = secondary zone of control
- `:zone` = authoritative zone for a given domain or subdomain

(**Note:** contrast `:zone` with `:domain` which simply represents the next level "up" in the hierarchy from the current domain).

Synapse uses the following logic for suffixes and zones upon `inet:fqdn` creation:

1. All domains consisting of a single element (such as `com`, `museum`, `us`, `br`, etc.) are considered **suffixes** and receive the following default values:
   - `:issuffix = 1`
   - `:iszone = 0`
   - `:zone = <none / property not created>`
   - `:domain = <none / property not created>`
2. Any domain whose **parent domain is a suffix** is considered a **zone** and receives the following default values:
   - `:issuffix = 0`
   - `:iszone = 1`
   - `:zone = <set to self>`
   - `:domain = <set to parent domain>`
3. Any domain whose **parent domain is a zone** is considered a "normal" subdomain and receives the following default values:
   - `:issuffix = 0`
   - `:iszone = 0`
   - `:zone = <set to parent domain>`
   - `:domain = <set to parent domain>`
4. Any domain whose parent domain is a "normal" subdomain receives the following default values:
   - `:issuffix = 0`
   - `:iszone = 0`
   - `:zone = <set to first fqdn "up" the domain hierarchy with :iszone = 1>`
   - `:domain = <set to parent domain>`

Note: The above logic is **recursive** over all nodes in a Cortex. Changing an `:issuffix` or `:iszone` property on an existing `inet:fqdn` node will not only modify that node, but also propagate any changes associated with those properties to any existing subdomains.

Potential Limitations

This logic works well for single-element top-level domains (TLDs) (such as `com` vs `microsoft.com`). However, it does not address cases that may be relevant for certain types of analysis, such as:

- **Top-level country code domains and their subdomains.** Under Synapse's default logic `uk` is a suffix and `co.uk` is a zone. However, `co.uk` could **also** be considered a suffix in its own right, because subdomains such as `somecompany.co.uk` are under the control of the organization that registers them. In this case, `uk` would be a suffix, `com.uk` could be considered both a suffix **and** a zone, and `somecompany.co.uk` could be considered a zone.
- **Special-case zones of control.** Some domains (such as those used to host web-based services) can be considered specialized zones of control. In these cases, the service provider typically owns the "main" domain (such as `wordpress.com`) but individual customers can register personal subdomains for their hosted services (such as `joesblog.wordpress.com`). The division between `wordpress.com` and individual customer subdomains could represent different zones of control. In this case, `com` would be a suffix, `wordpress.com` could be considered both a suffix **and** a zone, and `joesblog.wordpress.com` could be considered a zone.

Examples such as these are **not accounted for** by Synapse's suffix / zone logic. The definition of additional domains as suffixes and / or zones is an implementation decision (though once the relevant properties are set, the changes are propagated recursively as noted above).

Operations

Because of Synapse's reverse string indexing for `inet:fqdn` types, domains can be lifted or filtered based on matching any partial domain suffix string. The asterisk (`*`) is the extended operator used to perform this operation. The asterisk does **not** have to be used along dot boundaries but can match anywhere in any FQDN element.

Examples

Lift all domains that end with `yahooapis.com`:

```
storm> inet:fqdn='*yahooapis.com'
inet:fqdn=ayuisyahooapis.com
        :domain = com
        :host = ayuisyahooapis
        :issuffix = false
        :iszone = true
        :zone = ayuisyahooapis.com
        .created = 2024/04/19 14:27:10.137
inet:fqdn=micyuisyahooapis.com
        :domain = com
        :host = micyuisyahooapis
        :issuffix = false
        :iszone = true
        :zone = micyuisyahooapis.com
        .created = 2024/04/19 14:27:10.144
inet:fqdn=usyahooapis.com
        :domain = com
        :host = usyahooapis
        :issuffix = false
        :iszone = true
        :zone = usyahooapis.com
        .created = 2024/04/19 14:27:10.150
```

Lift all domains ending with `s.wordpress.com`:

```
storm> inet:fqdn="*s.wordpress.com"
inet:fqdn=s.wordpress.com
        :domain = wordpress.com
        :host = s
        :issuffix = false
        :iszone = false
        :zone = wordpress.com
        .created = 2024/04/19 14:27:10.209
inet:fqdn=dogs.wordpress.com
        :domain = wordpress.com
        :host = dogs
        :issuffix = false
        :iszone = false
        :zone = wordpress.com
        .created = 2024/04/19 14:27:10.203
inet:fqdn=sss.wordpress.com
        :domain = wordpress.com
        :host = sss
        :issuffix = false
        :iszone = false
        :zone = wordpress.com
        .created = 2024/04/19 14:27:10.216
inet:fqdn=www.sss.wordpress.com
        :domain = sss.wordpress.com
        :host = www
        :issuffix = false
        :iszone = false
        :zone = wordpress.com
        .created = 2024/04/19 14:27:10.216
inet:fqdn=cats.wordpress.com
        :domain = wordpress.com
        :host = cats
        :issuffix = false
        :iszone = false
        :zone = wordpress.com
        .created = 2024/04/19 14:27:10.195
```

Downselect a set of DNS A records to those with domains ending with `.museum`:

```
storm> inet:dns:a +:fqdn="*.museum"
inet:dns:a=('woot.museum', '5.6.7.8')
        :fqdn = woot.museum
        :ipv4 = 5.6.7.8
        .created = 2024/04/19 14:27:10.272
```

Usage Notes

- Because the asterisk is a non-alphanumeric character, the string to be matched must be enclosed in single or double quotes (see *Whitespace and Literals in Storm*).
- Because domains are reverse-indexed instead of prefix indexed, for **lift** operations, partial string matching can only occur based on the end (suffix) of a domain. It is not possible to **lift** FQDNs by prefix. For example, `inet:fqdn^=yahoo` is invalid.

- Domains can be **filtered** by prefix (^=). For example, `inet:fqdn="*.biz" +inet:fqdn^=smtp` is valid.
- Domains cannot be **filtered** based on suffix matching (note that a "lift by suffix" is effectively a combined "lift and filter" operation).
- Domains can be lifted or filtered using the regular expression (regex) extended operator (~=). For example `inet:fqdn~=google` is valid (see *Lift by Regular Expression (~=)* and *Filter by Regular Expression (~=)*).

inet:ipv4

IPv4 addresses are stored as integers and represented (displayed) to users as dotted-decimal strings.

Indexing

IPv4 addresses are indexed as integers. This optimizes various comparison operations, including greater than / less than, range, etc.

Parsing

While IPv4 addresses are stored and indexed as integers, they can be input into Storm (and used within Storm operations) as any of the following.

- integer: `inet:ipv4 = 3232235521`
- hex: `inet:ipv4 = 0xC0A80001`
- dotted-decimal string: `inet:ipv4 = 192.168.0.1`
- range: `inet:ipv4 = 192.168.0.1-192.167.0.10`
- CIDR: `inet:ipv4 = 192.168.0.0/24`

Insertion

The ability to specify IPv4 values using either range or CIDR format allows you to "bulk create" sets of `inet:ipv4` nodes without the need to specify each address individually.

Examples

Note: results (output) not shown below due to length.

Create ten `inet:ipv4` nodes:

```
[ inet:ipv4 = 192.168.0.1-192.168.0.10 ]
```

Create the 256 addresses in the range 192.168.0.0/24:

```
[ inet:ipv4 = 192.168.0.0/24 ]
```

Operations

Similar to node insertion, lifting or filtering IPV4 addresses by range or by CIDR notation will operate on every `inet:ipv4` node that exists within the Cortex and falls within the specified range or CIDR block. This allows operating on multiple contiguous IP addresses without the need to specify them individually.

Examples

Lift all `inet:ipv4` nodes within the specified range that exist within the Cortex:

```
storm> inet:ipv4 = 169.254.18.24-169.254.18.64
inet:ipv4=169.254.18.30
        :type = linklocal
        .created = 2024/04/19 14:27:10.460
inet:ipv4=169.254.18.36
        :type = linklocal
        .created = 2024/04/19 14:27:10.466
inet:ipv4=169.254.18.53
        :type = linklocal
        .created = 2024/04/19 14:27:10.471
```

Filter a set of DNS A records to only include those whose IPv4 value is within the 172.16.* RFC1918 range:

```
storm> inet:dns:a:fqdn=woot.com +:ipv4=172.16.0.0/12
inet:dns:a=('woot.com', '172.16.47.12')
        :fqdn = woot.com
        :ipv4 = 172.16.47.12
        .created = 2024/04/19 14:27:10.528
```

int

An `int` is an integer value. Synapse stores, indexes, and displays integer values as decimal integers, but will also accept as hex or octal values as input.

Indexing

N/A

Parsing

When adding or modifying integer values, Synapse will accept integer, hex (preceded by 0x), and octal (preceded by 0o) values and represent them as decimal integer values.

Examples

Set the `:count` of the `biz:bundle` to 42:

```
storm> biz:bundle=9688955d141aae88194277e74d82084d [ :count=42 ]
biz:bundle=9688955d141aae88194277e74d82084d
        :count = 42
        .created = 2024/04/19 14:27:10.573
```

Use a hex value to set the `:ip:proto` property for the `inet:flow` node to 6:

```
storm> inet:flow=684babd42810ae9dc11132805abc2831 [ :ip:proto=0x06 ]
inet:flow=684babd42810ae9dc11132805abc2831
        :dst = tcp://142.118.95.50
        :dst:ipv4 = 142.118.95.50
        :dst:proto = tcp
        :ip:proto = 6
        .created = 2024/04/19 14:27:10.620
```

Use an octal value to set the `:posix:perms` property for the `file:archive:entry` node to 755:

```
storm> file:archive:entry=3a24e1008b43bc2f1e35b3e872f201fc [ :posix:perms=0o426 ]
file:archive:entry=3a24e1008b43bc2f1e35b3e872f201fc
        :added = 2023/08/11 11:33:00.000
        :file = sha256:0c72088f529dc53e813de8e7df47922b1a9137924e072468559f7865eb7ad18b
        :posix:perms = 278
        :user = ozzie
        .created = 2024/04/19 14:27:10.666
```

Insertion

Same as for parsing.

Operations

Use integer, hex, or octal values to lift and filter integer types using standard comparison operators.

Examples

Lift all `risk:alert` nodes where the `:priority` is set to less than 10:

```
storm> risk:alert:priority<10
risk:alert=a592f0ee299c0000ce34cad33604d3b9
        :desc = outbound traffic to SOC-reported IP
        :detected = 2023/08/01 09:00:00.000
        :name = suspicious outbound traffic
        :priority = 8
        .created = 2024/04/19 14:27:10.714
```

Lift all `inet:flow` nodes tagged with `#my.tag` and filter to include only those where the `:ip:proto` property is set to the hex equivalent of 6:

```
storm> inet:flow#mytag +:ip:proto=0x06
```

Use an octal value to lift all `it:group` nodes where the `:posix:gid` values equate to 278:

```
storm> it:group:posix:gid=0o426
it:group=32076583262d2fd0e065812bc88723cf
        :desc = threat researchers group
        :host = a6f4147c23421ef47a1fabea899b3aeb
        :name = research
        :posix:gid = 278
        .created = 2024/04/19 14:27:10.822
```

ival

`ival` is a specialized type consisting of two `time` types in a paired `(<min>, <max>)` relationship. As such, the individual values in an `ival` are subject to the same specialized handling as individual *time* values.

`ival` types have their own optimizations in addition to those related to `time` types.

Indexing

N/A

Parsing

An `ival` type is typically specified as two comma-separated time values enclosed in parentheses. Alternately, an `ival` can be specified as a single time value with no parentheses (see **Insertion** below for `ival` behavior when specifying a single time value).

Single or double quotes are required in accordance with the standard rules for using *Whitespace and Literals in Storm*. For example:

- `.seen=("2017/03/24 12:13:27", "2017/08/05 17:23:46")`
- `+#sometag=(2018/09/15, "+24 hours")`
- `.seen=2019/03/24`

As `ival` types are a pair of values (i.e., an explicit minimum and maximum), the values must be placed in parentheses and separated by a comma: `(<min>, <max>)`. The parser expects two **explicit** values.

An `ival` can also be specified as a single time value, in which case the value must be specified **without** parentheses: `<time>`. See **Insertion** below for `ival` behavior when adding vs. modifying using a single time value vs. a `(<min>, <max>)` pair.

When entering an `ival` type, each time value can be input using most of the acceptable formats for *time* types, including explicit times (including lower resolution times and wildcard times), relative times, and the special values `now` and `?`.

`ival` types also support relative times using `+-` format to represent both a positive and negative offset from a given point (i.e., `"+-1 hour"`).

When entering relative times in an `ival` type:

- A relative time in the **first** (`<min>`) position is calculated relative to the **current time** (`now`).
- A relative time in the **second** (`<max>`) position is calculated relative to the **first** (`<min>`) time.

For example:

- `.seen="+1 hour"` means from the current time (now) to one hour after the current time.
- `.seen=(2018/12/01, "+1 day")` means from 12:00 AM December 1, 2018 to 12:00 AM December 2, 2018.
- `.seen=(2018/12/01, "-1 day")` means from 12:00 AM November 30, 2018 to 12:00 AM December 1, 2018.
- `.seen=(now, "+-5 minutes")` means from 5 minutes ago to 5 minutes from now.
- `.seen=("-30 minutes", "+1 hour")` means from 30 minutes ago to 30 minutes from now.

When specifying minimum and maximum times for an `ival` type (or when specifying minimum and maximum `time` values to the `*range=` comparator), the following restrictions should be kept in mind:

- Minimums and maximums that use explicit times and / or special terms (`now`, `?`) should be specified in `<min>, <max>` order.
  - Specifying a `<max>, <min>` order will **not** result in an error message, but because it results in an exclusionary time window, it will not return any nodes (i.e., no time / interval can be both greater than a max value and less than a min value).
  - Similarly, combinations of relative times that result in an effective `<max>, <min>` after relative offsets are calculated are allowed (will not generate an error), but will result in an exclusionary time window that does not return any nodes.
- Values that result in a nonsensical `<min>, <max>` are not allowed and will generate an error. For example:
  - The special value `?` cannot be used as a minimum value in a `(<min>, <max>)` pair.
  - A `+-` relative time cannot be used as a minimum value in a `(<min>, <max>)` pair.
  - When specifying a `+-` relative time as the maximum value in a `(<min>, <max>)` pair, an explicit `<min>` value is also required (i.e., either an explicit time or `now`).

Insertion

- When **adding** an `ival` as a `(<min>, <max>)` pair, the `ival` can be specified as described above.
  - If the values for `<min>` and `<max>` are identical, then `<min>` will be set to the specified value and `<max>` will be set to `<min>` plus 1 ms.
- When **adding** an `ival` as a single time value, it must be specified **without** parentheses.
  - When a single time value is used, the `<min>` value will be set to the specified time and the `<max>` will be set to the `<min>` time plus 1 ms.
- When **modifying** an existing `ival` property (including tag timestamps) with either a `(<min>, <max>)` pair or a single time value, the existing `ival` is **not** simply overwritten (as is the norm for modifying properties - see *Storm Reference - Data Modification*). Instead, the `<min>` and / or `<max>` are **only** updated if the new value(s) are:
  - Less than the current `<min>`, and / or
  - Greater than the current `<max>`.

  This means that once set, `<min>` and `<max>` can only be "pushed out" to a lower minimum and / or a higher maximum. Specifying a time or times that fall **within** the current minimum and maximum will have no effect (i.e., the current values will be retained).

  This means that it is not possible to "shrink" an `ival` directly; to specify a higher minimum or a lower maximum (or to remove the timestamps altogether), you must delete the `ival` property (or remove the timestamped tag) and re-add it with the updated values.

Operations

`ival` types can be lifted and filtered (see *Storm Reference - Lifting* and *Storm Reference - Filtering*) with the standard equivalent (=) operator, which will match the **exact** `<min>` and `<max>` values specified.

Example:

Lift the DNS A nodes whose observation window is **exactly** from 2018/12/13 01:05 to 2018/12/16 12:57:

```
storm> inet:dns:a.seen=("2018/12/13 01:05", "2018/12/16 12:57")
inet:dns:a=('yoyodyne.com', '16.16.16.16')
        :fqdn = yoyodyne.com
        :ipv4 = 16.16.16.16
        .created = 2024/04/19 14:27:10.877
        .seen = ('2018/12/13 01:05:00.000', '2018/12/16 12:57:00.000')
```

`ival` types cannot be used with comparison operators such as "less than" or "greater than or equal to".

`ival` types are most often lifted or filtered using the custom interval comparator (`@=`) (see *Lift by Time or Interval (@=)* and *Filter by Time or Interval (@=)*). `@=` is intended for time-based comparisons (including comparing `ival` types with `time` types).

Example:

Lift all the DNS A nodes whose observation window overlaps with the interval of March 1, 2019 through April 1, 2019:

```
storm> inet:dns:a.seen@=(2019/03/01, 2019/04/01)
inet:dns:a=('hurr.com', '4.4.4.4')
        :fqdn = hurr.com
        :ipv4 = 4.4.4.4
        .created = 2024/04/19 14:27:10.924
        .seen = ('2019/01/05 09:38:00.000', '2019/03/12 18:17:00.000')
inet:dns:a=('derp.net', '8.8.8.8')
        :fqdn = derp.net
        :ipv4 = 8.8.8.8
        .created = 2024/04/19 14:27:10.932
        .seen = ('2019/03/08 07:26:00.000', '2019/03/22 10:14:00.000')
inet:dns:a=('blergh.org', '2.2.2.2')
        :fqdn = blergh.org
        :ipv4 = 2.2.2.2
        .created = 2024/04/19 14:27:10.939
        .seen = ('2019/03/28 22:22:00.000', '2019/04/27 00:03:00.000')
```

`ival` types cannot be used with the `*range=` custom comparator. `*range=` can only be used to specify a range of individual values (such as `time` or `int`).

loc

`Loc` is a specialized type used to represent geopolitical locations (i.e., locations within geopolitical boundaries) as a series of user-defined dot-separated hierarchical strings - for example, *<country>.<state / province>.<city>*. This allows specifying locations such as `us.fl.miami`, `gb.london`, and `ca.on.toronto`.

`Loc` is an extension of the *str* type. However, because `loc` types use strings that comprise a dot-separated hierarchy, they exhibit slightly modified behavior from standard string types for certain operations.

Indexing

The loc type is an extension of the *str* type and so is **prefix-indexed** like other strings. However, the use of dot-separated boundaries impacts operations using loc values.

loc values are normalized to lowercase.

Parsing

loc values can be input using any case (uppercase, lowercase, mixed case) but will normalized to lowercase.

Components of a loc value must be separated by the dot (.) character. The dot is a reserved character for the loc type and is used to separate string elements along hierarchical boundaries. The use of the dot as a reserved boundary marker impacts operations using the loc type. Note that this means the dot cannot be used as part of a location string. For example, the following location value would be interpreted as a hierarchical location with four elements (us, fl, st, and petersburg):

- `:loc = us.fl.st.petersburg`

To appropriately represent the "city" element of the above location, an alternate syntax must be used. For example:

- `:loc = us.fl.stpetersburg`
- `:loc = "us.fl.saint petersburg"`
- …etc.

As an extension of the str type, loc types are subject to Synapse's restrictions regarding using *Whitespace and Literals in Storm*.

Insertion

Same as for parsing.

As loc values are simply dot-separated strings, the use or enforcement of any specific convention for geolocation values and hierarchies is an implementation decision.

Operations

The use of the dot character (.) as a reserved boundary marker impacts prefix (^=) and equivalent (=) operations using the loc type.

String and string-derived types are **prefix-indexed** to optimize lifting or filtering strings that start with a given substring using the prefix (^=) extended comparator. For standard strings, the prefix comparator can be used with strings of arbitrary length. However, for string-derived types (including loc) that use dotted hierarchical notation, **the prefix comparator operates along dot boundaries.**

This is because the analytical significance of a location string is likely to fall on these hierarchical boundaries as opposed to an arbitrary substring prefix match. That is, it is more likely to be analytically meaningful to lift all locations within the US (^=us) or within Florida (^=us.fl) than it is to lift all locations in the US within states that start with "V" (^=us.v).

Prefix comparison for loc types is useful because it easily allows lifting or filtering at any appropriate level of resolution within the dotted hierarchy:

Examples:

Lift all organizations with locations in Turkey:

```
storm> ou:org:loc^=tr
ou:org=67aa10b608319cc7c3283a91fad7fe09
        :loc = tr.ankara
        :name = republic of turkey ministry of foreign affairs
        .created = 2024/04/19 14:27:10.987
ou:org=48f00d29571731e25b4bf6ef6cbbcfc2
        :loc = tr.istanbul
        :name = adeo it consulting services
        .created = 2024/04/19 14:27:10.993
```

Lift all IP addresses geolocated in the province of Ontario, Canada:

```
storm> inet:ipv4:loc^=ca.on
inet:ipv4=149.248.52.240
        :loc = ca.on
        :type = unicast
        .created = 2024/04/19 14:27:11.037
inet:ipv4=49.51.12.195
        :loc = ca.on.barrie
        :type = unicast
        .created = 2024/04/19 14:27:11.045
inet:ipv4=199.201.123.200
        :loc = ca.on.keswick
        :type = unicast
        .created = 2024/04/19 14:27:11.050
```

Note: Specifying a more granular prefix value will **not** match values that are less granular. That is `:loc^=ca.on` will fail to match `:loc=ca`.

Lift all places in the city of Seattle:

```
storm> geo:place:loc=us.wa.seattle
geo:place=0dadfb545126166d5e332a959e812b2d
        :latlong = 47.4502535,-122.3110105
        :loc = us.wa.seattle
        :name = seattle-tacoma international airport
        .created = 2024/04/19 14:27:11.104
geo:place=981bddc8f063b58cdce8dc16c6603b9a
        :latlong = 47.6205099,-122.3514714
        :loc = us.wa.seattle
        :name = space needle
        .created = 2024/04/19 14:27:11.096
```

Usage Notes

- Use of the equals comparator (=) with `loc` types will match the **exact value only.** So `:loc = us` will match **only** `:loc = us` but not `:loc = us.ca` or `:loc = us.il.chicago`.
- Because the prefix match operates on the dot boundary, attempting to lift or filter by a prefix string match that does **not** fall on a dot boundary will not return any nodes. For example, the filter syntax `+:loc^=us.v` will fail to return any nodes even if nodes with `:loc = us.vt` or `:loc = us.va` exist. (However, it would return nodes with `:loc = us.v` or `:loc = us.v.foo` if such nodes exist.)

str

Indexing

String (and string-derived) types are indexed by **prefix** (character-by-character from the beginning of the string). This allows matching on any initial substring.

Parsing

Some string types and string-derived types are normalized to all lowercase to facilitate pivoting across like values without case-sensitivity. For types that are normalized in this fashion, the string can be entered in mixed-case and will be automatically converted to lowercase.

Strings are subject to Synapse's restrictions regarding using *Whitespace and Literals in Storm*.

Insertion

Same as for parsing.

Operations

Because of Synapse's use of **prefix indexing,** string and string-derived types can be lifted or filtered based on matching an initial substring of any string using the prefix extended comparator (^=) (see *Lift by Prefix (^=)* and *Filter by Prefix (^=)*).

Prefix matching is case-sensitive based on the specific type being matched. If the target property's type is case-sensitive, the string to match must be entered in case-sensitive form. If the target property is case-insensitive (i.e., normalized to lowercase) the string to match can be entered in any case (upper, lower, or mixed) and will be automatically normalized by Synapse.

Examples

Lift all organizations whose name starts with the word "Acme ":

```
storm> ou:org:name^='acme '
ou:org=bbb36f4645f031e103b6931b87f18da5
        :name = acme construction
        :names = ['acme construction']
        .created = 2024/04/19 14:27:09.538
ou:org=1c6bda5011ba158abab4d6decd73698b
        :name = acme consulting
        :names = ['acme consulting']
        .created = 2024/04/19 14:27:09.532
```

Filter a set of Internet accounts to those with usernames starting with 'matrix':

```
storm> inet:web:acct:site=twitter.com +:user^=matrix
inet:web:acct=twitter.com/matrixneo
        :site = twitter.com
        :user = matrixneo
        .created = 2024/04/19 14:27:11.178
inet:web:acct=twitter.com/matrixmaster
```

(continues on next page)

(continued from previous page)

```
        :site = twitter.com
        :user = matrixmaster
        .created = 2024/04/19 14:27:11.170
```

Strings and string-derived types can also be lifted or filtered using the regular expression extended comparator (~=) (see *Lift by Regular Expression (~=)* and *Filter by Regular Expression (~=)*).

syn:tag

`syn:tag` is a specialized type used for *Tag* nodes within Synapse. Tags represent domain-specific, analytically relevant observations or assessments. They support a hierarchical namespace based on user-defined dot-separated strings. This hierarchy allows recording classes or categories of analytical observations that can be defined with increasing specificity. (See *Analytical Model* for more information.)

`syn:tag` is an extension of the *str* type. However, because `syn:tag` types use strings that comprise a dot-separated hierarchy, they exhibit slightly modified behavior from standard string types for certain operations.

Indexing

The `syn:tag` type is an extension of the *str* type and so is **prefix-indexed** like other strings. However, the use of dot-separated boundaries impacts some operations using `syn:tag` values.

`syn:tag` values are normalized to lowercase.

Parsing

`syn:tag` values can contain lowercase characters, numerals, and underscores. Spaces and ASCII symbols (other than the underscore) are not allowed. If you attempt to create a tag name that includes a dash character (`-`) it will automatically be converted to an underscore (`_`).

Note: Synapse includes support for Unicode words in tag strings; this includes most characters that can be part of a word in any language.

Components of a `syn:tag` value must be separated by the dot (`.`) character. The dot is a reserved character for the `syn:tag` type and is used to separate string elements along hierarchical boundaries. The use of the dot as a reserved boundary marker impacts some operations using the `syn:tag` type.

`syn:tag` values can be input using any case (uppercase, lowercase, mixed case) but will be normalized to lowercase. As noted above, dashes are automatically converted to underscores.

As `syn:tag` values cannot contain whitespace (spaces) or escaped characters, the Synapse restrictions regarding using *Whitespace and Literals in Storm* do **not** apply.

Examples

The following are all allowed `syn:tag` values:

- `syn:tag = rep.vt.exploit`
- `syn:tag = aka.kaspersky.mal.shamoon.2`
- `syn:tag = cno.tgt.cn_mil_pla`

The following `syn:tag` values are not allowed and will generate `BadTypeValu` errors:

- `syn:tag = this.is.my.@#$*(.tag` (contains disallowed characters)
- `syn:tag = "some.threat group.tag"` (contains whitespace)

Insertion

A `syn:tag` node does not have to be created before the equivalent tag can be applied to another node. That is, applying a tag to a node will result in the automatic creation of the corresponding `syn:tag` node or nodes (assuming the appropriate user permissions). For example:

```
storm> [inet:fqdn=woot.com +#some.new.tag ]
inet:fqdn=woot.com
        :domain = com
        :host = woot
        :issuffix = false
        :iszone = true
        :zone = woot.com
        .created = 2024/04/19 14:27:10.264
        #some.new.tag
```

The above Storm syntax will both apply the tag `#some.new.tag` to the node `inet:fqdn = woot.com` and automatically create the node `syn:tag = some.new.tag` if it does not already exist (as well as `syn:tag = some` and `syn:tag = some.new`). This behavior (based on creating the FQDN `woot.com` and applying the tag `#some.new.tag` in the previous example) is shown below by lifting tags that begin with 'some':

```
storm> syn:tag^=some
syn:tag=some
        :base = some
        :depth = 0
        .created = 2024/04/19 14:27:11.227
syn:tag=some.new
        :base = new
        :depth = 1
        :up = some
        .created = 2024/04/19 14:27:11.227
syn:tag=some.new.tag
        :base = tag
        :depth = 2
        :up = some.new
        .created = 2024/04/19 14:27:11.227
```

Operations

The use of the dot character (`.`) as a reserved boundary marker impacts prefix (`^=`) and equivalent (=) operations using the `syn:tag` type.

String and string-derived types are **prefix-indexed** to optimize lifting or filtering strings that start with a given substring using the prefix (`^=`) extended comparator. For standard strings, the prefix comparator can be used with strings of arbitrary length. However, for string-derived types (including `syn:tag`) that use dotted hierarchical notation, **the prefix comparator operates along dot boundaries.**

This is because the analytical significance of a tag is likely to fall on these hierarchical boundaries as opposed to an arbitrary substring prefix match. That is, it is more likely to be analytically meaningful to lift all nodes with that are related to sinkhole infrastructure (`syn:tag^=cno.infra.anon.sink`) than it is to lift all nodes with infrastructure tags that begin with "s" (`syn:tag^=cno.infra.anon.s`).

Prefix comparison for `syn:tag` types is useful because it easily allows lifting or filtering at any appropriate level of resolution within a tag hierarchy:

Lift all tags in the computer network operations (`cno`)tree:

```
storm> syn:tag^=cno
syn:tag=cno
        :base = cno
        :depth = 0
        .created = 2024/04/19 14:27:11.271
syn:tag=cno.mal
        :base = mal
        :depth = 1
        :up = cno
        .created = 2024/04/19 14:27:11.277
syn:tag=cno.mal.redtree
        :base = redtree
        :depth = 2
        :up = cno.mal
        .created = 2024/04/19 14:27:11.277
syn:tag=cno.threat
        :base = threat
        :depth = 1
        :up = cno
        .created = 2024/04/19 14:27:11.271
syn:tag=cno.threat.t27
        :base = t27
        :depth = 2
        :up = cno.threat
        .created = 2024/04/19 14:27:11.271
```

Lift all tags representing aliases (e.g., names of malware, threat groups, etc.) reported by Symantec:

```
storm> syn:tag^=aka.symantec
syn:tag=aka.symantec
        :base = symantec
        :depth = 1
        :up = aka
        .created = 2024/04/19 14:27:11.320
syn:tag=aka.symantec.mal
        :base = mal
        :depth = 2
        :up = aka.symantec
        .created = 2024/04/19 14:27:11.320
syn:tag=aka.symantec.mal.bifrose
        :base = bifrose
        :depth = 3
        :up = aka.symantec.mal
        .created = 2024/04/19 14:27:11.320
syn:tag=aka.symantec.thr
        :base = thr
        :depth = 2
```

(continues on next page)

(continued from previous page)

```
        :up = aka.symantec
        .created = 2024/04/19 14:27:11.326
syn:tag=aka.symantec.thr.cadelle
        :base = cadelle
        :depth = 3
        :up = aka.symantec.thr
        .created = 2024/04/19 14:27:11.326
```

Lift all tags representing anonymous VPN infrastructure:

```
storm> syn:tag^=cno.infra.anon.vpn
syn:tag=cno.infra.anon.vpn
        :base = vpn
        :depth = 3
        :up = cno.infra.anon
        .created = 2024/04/19 14:27:11.370
syn:tag=cno.infra.anon.vpn.airvpn
        :base = airvpn
        :depth = 4
        :up = cno.infra.anon.vpn
        .created = 2024/04/19 14:27:11.370
syn:tag=cno.infra.anon.vpn.nordvpn
        :base = nordvpn
        :depth = 4
        :up = cno.infra.anon.vpn
        .created = 2024/04/19 14:27:11.376
```

Note that specifying a more granular prefix value will **not** match values that are less granular. That is, `syn:tag^=cno.infra` will fail to match `syn:tag = cno`.

Similarly, use of the equals comparator (=) with `syn:tag` types will match the **exact value only.** So `syn:tag = aka` will match **only** that tag but not `syn:tag = aka.symantec` or `syn:tag = aka.trend.thr.pawnstorm`.

Because the prefix match operates on the dot boundary, attempting to lift or filter by a prefix string match that does **not** fall on a dot boundary will not return any nodes. For example, the syntax `syn:tag^=aka.t` will fail to return any nodes even if nodes `syn:tag = aka.talos` or `syn:tag = aka.trend` exist. (However, it would return nodes `syn:tag = aka.t` or `syn:tag = aka.t.foo` if such nodes exist.)

time

Synapse stores `time` types in Epoch milliseconds (millis) - that is, the number of milliseconds since January 1, 1970. The `time` type is technically a date/time because it encompasses both a date and a time. A time value alone, such as 12:37 PM (12:37:00.000), is invalid.

See also the section on *ival* (interval) types for details on how `time` types are used as minimum / maximum pairs.

Indexing

N/A

Parsing

`time` values can be input into Storm as any of the following:

- **Explicit** times:
  - Human-readable (YYYY/MM/DD hh:mm:ss.mmm):

    `"2018/12/16 09:37:52.324"`
  - Human-readable "Zulu" (YYYY/MM/DDThh:mm:ss.mmmZ):

    `2018/12/16T09:37:52.324Z`
  - Human-readable with time zone (YYYY-MM-DD hh:mm:ss.mmm+/-hh:mm). No spaces are allowed between the time value and the time zone offset:

    `2018-12-16 09:37:52.324-04:00`

    Note: Synapse does not support the **storage** of an explicit time zone with a time value (i.e., +0800). Synapse stores time values in UTC for consistency. If a time zone is specified using an acceptable time zone offset format on input, Synapse will automatically convert the value to UTC for storage. If no time zone is specified, Synapse will assume the value is in UTC.
  - No formatting (YYYYMMDDhhmmssmmm):

    `20181216093752324`
  - Epoch millis:

    `(1544953072324)`

    Note: Synapse expects time values to be entered as parseable time **strings** (such as 2018/12/16 09:37:52.324, which Synapse internally converts to a millis integer for storage). To enter a time in raw epoch millis format, you must enclose it in parentheses so that Synapse interprets the value as a raw integer. (Otherwise, Synapse will attempt to interpret the value as a "no formatting" string, and throw an error.)

- **Relative** (offset) time values in the format:

  + | - | +- <*count*> <*unit*>

  where <*count*> is a numeric value and <*unit*> is one of the following:
  - `minute(s)`
  - `hour(s)`
  - `day(s)`

  Examples:
  - `"+7 days"`
  - `"-15 minutes"`

  - `"+-1 hour"`
- **"Special"** time values:
  - the keyword `now` is used to represent the current date/time.
  - a question mark (`?`) is used to effectively represent an unspecified / indefinite time in the future (technically equivalent to 9223372036854775807 millis, i.e., "some really high value that is probably the heat death of the universe". Note that technically the largest valid millis value is 9999999999999 (thirteen 9's), which represents 2286/11/20 09:46:39.999).

    The question mark can be used as the maximum value of an interval (*ival*) type to specify that the data or assessment associated with the `ival` should be considered valid indefinitely. (Contrast that with a maximum interval value set to the equivalent of `now` that would need to be continually updated over time in order to remain current.)

Standard rules regarding using *Whitespace and Literals in Storm* apply. For example, `"2018/12/16 09:37:52.324"` needs to be entered in single or double quotes, but `2018/12/16` does not. Similarly, relative times starting with `+` or `-` and the special time value `?` need to be placed in single or double quotes.

Lower Resolution Time Values and Wildcard Time Values

`time` values (including tag timestamps) must be entered at a minimum resolution of year (`YYYY`) and can be entered up to a maximum resolution of milliseconds (`YYYY/MM/DD hh:mm:ss.mmm`).

Where lower resolution values are entered, Synapse will make logical assumptions about the intended date / time value and zero-fill the remainder of the equivalent epoch mills date / time. For example:

- A value of `2016` will be interpreted as 12:00 AM on January 1, 2016 (`2016/01/01 00:00:00.000`).
- A value of `2018/10/27` will be interpreted as 12:00 AM on that date (`2018/10/27 00:00:00.000`).
- A value of `"2020/03/16 05"` will be interpreted as 05:00 AM on that date (`2020/03/16 05:00:00.000`).
- A value of `"2018/10/27 14:00-04:00"` will be interpreted as 14:00 (2:00 PM) on that date with a 4 hour offset from UTC (`2018/10/27 14:00:00.000-04:00`, stored in UTC as `2018/10/27 18:00:00.000`).

Synapse also supports the use of the wildcard (`*`) character to specify a partial time value match:

- A value of `2016*` will be interpreted as "any date / time within the year 2016".
- A value of `2018/10/27*` will be interpreted as "any time on October 27, 2018".
- A value of `"2020/03/16 05*"` will be interpreted as "any time within the hour of 05:00 on March 16, 2020".

Note: When using wildcard syntax, the wildcard must be used on a sensible time value boundary, such as `YYYYMM*`. You cannot us a wildcard to "split" values (i.e., `YYMMD*` is invaild syntax).

Examples:

Set the time of a DNS request to the current time:

```
storm> [ inet:dns:request="*" :query:name=woot.com :time=now ]
inet:dns:request=f926b06d72d49c9eb4c25ade94250dea
        :query:name = woot.com
        :query:name:fqdn = woot.com
        :time = 2024/04/19 14:27:11.425
        .created = 2024/04/19 14:27:11.424
```

Set the observed time window (technically an `ival` type) for when an IP address was a known sinkhole (via the `#cno.infra.dns.sink.hole` tag) from its known start date to an indefinite future time (i.e., the sinkhole is presumed to remain a sinkhole indefinitely / until the values are manually updated with an explicit end date):

```
storm> [ inet:ipv4=1.2.3.4 +#cno.infra.dns.sink.hole=(2017/06/13, "?") ]
inet:ipv4=1.2.3.4
        :type = unicast
        .created = 2024/04/19 14:27:10.264
        #cno.infra.dns.sink.hole = (2017/06/13 00:00:00.000, ?)
```

- Set the observed time window using a time zone offset:

```
storm> [ inet:ipv4=5.6.7.8 +#cno.infra.dns.sink.hole=(2017/06/13 09:46+04:00, "?") ]
inet:ipv4=5.6.7.8
        :type = unicast
        .created = 2024/04/19 14:27:10.272
        #cno.infra.dns.sink.hole = (2017/06/13 05:46:00.000, ?)
```

Insertion

When adding or modifying `time` types, any of the above formats (explicit / relative / special terms) can be specified.

In addition, when adding or modifying `time` types, a lower resolution time and a wildcard time behave identically. In other words, the following are equivalent Storm queries (both will set the `:time` value of the newly created DNS request node to `2021/01/23 00:00:00.000`):

```
[ inet:dns:request="*" :time=2021/01/23 ]

[ inet:dns:request="*" :time=2021/01/23* ]
```

When specifying a relative time for a `time` value, **the offset will be calculated from the current time** (`now`):

```
storm> [ inet:dns:request="*" :query:name=woot.com :time="-5 minutes" ]
inet:dns:request=52a8775476692dc2e0301f20f944278c
        :query:name = woot.com
        :query:name:fqdn = woot.com
        :time = 2024/04/19 14:22:11.553
        .created = 2024/04/19 14:27:11.552
```

Plus / minus (`+-`) relative times cannot be specified for `time` types, as the type requires a single value. See the section on *ival* (interval) types for details on using `+-` times with `ival` types.

Operations

`time` types can be lifted and filtered using:

- Standard logical and mathematical comparison operators (comparators).
- The extended range (`*range=`) custom comparator.
- The extended interval (`@=`) custom comparator.

Standard Operators

`time` types can be lifted and filtered with the standard logical and mathematical comparators (see *Storm Reference - Lifting* and *Storm Reference - Filtering*). This includes the use of lower resolution time values and wildcard time values.

Example:

Downselect a set of DNS request nodes to those that occurred prior to June 1, 2019:

```
storm> inet:dns:request +:time<2019/06/01
inet:dns:request=6d568286a1fea38e079184db59a0313a
        :query:name = derp.net
        :query:name:fqdn = derp.net
        :time = 2015/12/14 19:22:00.000
        .created = 2024/04/19 14:27:11.583
inet:dns:request=80af11d3af2592df06d7d28366ffff39
        :query:name = hurr.com
        :query:name:fqdn = hurr.com
        :time = 2018/06/28 17:43:00.000
        .created = 2024/04/19 14:27:11.576
```

Note: It is important to understand the differences in behavior when lifting and filtering `time` types using lower resolution time values (which Synpase zero-fills) or wildcard time values (which Synpase wildcard-matches). These behaviors vary based on the specific operator used.

- When lifting or filtering using the equivalent (=) operator, behavior is **different:**
  - `:time=2021/05/13` means equal to **the exact date/time value** `2021/05/13 00:00:00.000`.
  - `:time=2021/05/13*` means equal to **any** time on that date (`2021/05/13 00:00:00.000` through `2021/05/13 23:59:59.999`).
- When lifting or filtering using the greater than (>) / greater than or equal to (>=) operators, behavior is **equivalent:**
  - `:time>2021/05/13` and `:time>2021/05/13*` **both** mean any date / time greater than `2021/05/13 00:00:00.000`.
  - `:time>=2021/05/13` and `:time>=2021/05/13*` **both** mean any date / time greater than or equal to `2021/05/13 00:00:00.000`.

  Both are equivalent because in this case Synapse interprets the wildcard syntax as "greater than or equal to the **lowest** possible wildcard match", which in this case is `2021/05/13 00:00:00.000`.
- When lifting or filtering using the less than (<) / less than or equal to (<=) operators, behavior is **different:**
  - `:time<2021/05/13` / `:time<=2021/05/13` mean any date / time less than (or less than or equal to) `2021/05/13 00:00:00.000`.
  - `:time<2021/05/13*` / `:time<=2021/05/13*` both mean any date / time less than (or less than or equal to) `2021/05/13 23:59:59.999`.

  The behavior differs because in this case Synapse interprets the wildcard syntax as "less than or equal to the **highest** possible wildcard match", which in this case is `2021/05/13 23:59:59.999`.

Tip: The wildcard syntax is useful because it can provide a simplified, more intuitive means to specify certain time ranges / time intervals without needing to use the range (`*range=`) or interval (`@=`) operators. For example, the following three Storm queries are equivalent and will return all files compiled at any time within the year 2019:

```
file:bytes:mime:pe:compiled=2019*

file:bytes:mime:pe:compiled*range=('2019/01/01 00:00:00.000', '2019/12/31 23:59:59.999')

file:bytes:mime:pe:compiled@=('2019/01/01', '2020/01/01')
```

(A **range** maximum value represents "less than or equal to" that value, while an **interval** maximum value represents "less than" that value.)

Range Custom Operator

`time` types can lifted and filtered using the `*range=` custom comparator (see *Lift by Range (*range=)* and *Filter by Range (*range=)*).

Example:

Lift a set of `file:bytes` nodes whose PE compiled time is between January 1, 2019 and today:

```
storm> file:bytes:mime:pe:compiled*range=(2019/01/01, now)
file:bytes=sha256:9f9d96e99cef99cbfe8d02899919a7f7220f2273bb36a084642f492dd3e473da
        :mime:pe:compiled = 2019/10/07 12:42:45.000
        :sha256 = 9f9d96e99cef99cbfe8d02899919a7f7220f2273bb36a084642f492dd3e473da
        .created = 2024/04/19 14:27:11.628
file:bytes=sha256:bd422f912affcf6d0830c13834251634c8b55b5a161c1084deae1f9b5d6830ce
        :mime:pe:compiled = 2021/04/13 00:23:14.000
        :sha256 = bd422f912affcf6d0830c13834251634c8b55b5a161c1084deae1f9b5d6830ce
        .created = 2024/04/19 14:27:11.634
```

Note: Both lower resolution times and wildcard times can be used for values specified within the `*range=` operator. Because the range operator is a shorthand syntax for "greater than or equal to <*range_min*> and less than or equal to <*range_max*>", users should be aware of differences in behavior between each kind of time value with greater than / less than operators.

See the Storm documents referenced above for additional examples using the range (`*range=`) comparator.

Interval Custom Operator

`time` types can be lifted and filtered using the interval (`@=`) custom comparator (see *Lift by Time or Interval (@=)* and *Filter by Time or Interval (@=)*). The comparator is specifically designed to compare `time` types and `ival` types, which can be useful (for example) for filtering to a set of nodes whose `time` properties fall within a specified interval.

Example:

Lift a set of DNS A records whose window of observation includes March 16, 2019 at 13:00 UTC:

```
storm> inet:dns:a.seen@='2019/03/16 13:00'
inet:dns:a=('aaaa.org', '1.2.3.4')
        :fqdn = aaaa.org
        :ipv4 = 1.2.3.4
        .created = 2024/04/19 14:27:11.680
```

(continues on next page)

(continued from previous page)

```
        .seen = ('2018/12/29 12:36:27.000', '2019/06/03 18:14:33.000')
inet:dns:a=('derp.net', '8.8.8.8')
        :fqdn = derp.net
        :ipv4 = 8.8.8.8
        .created = 2024/04/19 14:27:10.932
        .seen = ('2019/03/08 07:26:00.000', '2019/03/22 10:14:00.000')
inet:dns:a=('bbbb.edu', '5.6.7.8')
        :fqdn = bbbb.edu
        :ipv4 = 5.6.7.8
        .created = 2024/04/19 14:27:11.687
        .seen = ('2019/03/16 12:59:59.000', '2019/03/16 13:01:01.000')
```

Note: Both lower resolution times and wildcard time can be used for values specified within the `@=` operator. Because the interval operator is a shorthand syntax for "greater than or equal to <*ival_min*> and less than <*ival_max*>", users should be aware of differences in behavior between each kind of time value with greater than / less than operators.

See the Storm documents referenced above for additional examples using the interval (`@=`) comparator.

3.6.10 Storm Reference - Storm Commands

Storm commands are built-in or custom commands that can be used natively within Synapse Storm queries.

Built-in commands are native to the Storm library and loaded by default within a given Cortex. Built-in commands comprise a set of helper commands that perform a variety of specialized tasks that are useful regardless of the types of data stored in Synapse or the types of analysis performed.

Custom commands are Storm commands that have been added to a Cortex to invoke the execution of dynamically loaded modules. Synapse **Power-Ups** (*Power-Up*) are examples of modules that may install additional Storm commands to implement functionality specific to that Power-Up (such as querying a third-party data source to automatically ingest and model the data in Synapse).

Storm Commands and the Pipe Character

The pipe character (`|`) is used with Storm commands to:

- Return to Storm query syntax after running a Storm command.
- Separate individual Storm commands and their parameters (i.e., if you are "chaining" multiple commands together).

For example:

```
inet:fqdn=woot.com nettools.whois | nettools.dns --type A AAAA NS | -> inet:dns:a
```

The query above:

- lifts the FQDN `woot.com`,
- performs a live "whois" lookup using the Synapse-Nettools *Power-Up*,
- performs a live DNS query for the FQDN's A, AAAA, and NS records, and
- pivots from the FQDN to any associated DNS A records.

The pipe is used:

- to separate the two `nettools.*` commands, and
- to separate the `nettools.dns` command and its switches from the subsequent query operation (the pivot).

Tip: A pipe character is **not** required between a Storm operation and any **initial** Storm command (e.g., between the `inet:fqdn=woot.com` lift operation and the subsequent `nettools.whois` command in the example above). A pipe character can **optionally** be placed in this location (some users may find it easier to remember the "rules" for pipe use as "place a pipe between Storm operations and Storm commands"), but is not necessary.

Storm Command Reference

The full list of Storm commands (built-in and custom) available in a given instance of Synapse can be displayed with the `help` command.

Help for a specific Storm command can be displayed with `<command> --help`.

Tip: This section details the usage and syntax for **built-in** Storm commands. Many of the commands below - such as `count`, `intersect`, `limit`, `max` / `min`, `uniq`, or the various `gen` (generate) commands - directly support analysis tasks.

Other commands, such as those used to manage daemons, queues, packages, or services, are likely of greater interest to Synapse administrators or developers.

- *help*
- *aha*
- *auth*
- *background*
- *batch*
- *copyto*
- *cortex.httpapi*
- *count*
- *cron*
- *delnode*
- *diff*
- *divert*
- *dmon*
- *edges*
- *feed*
- *gen*
- *graph*
- *iden*
- *intersect*
- *layer*

- *lift*
- *limit*
- *macro*
- *max*
- *merge*
- *min*
- *model*
- *movenodes*
- *movetag*
- *nodes*
- *note*
- *once*
- *parallel*
- *pkg*
- *ps*
- *queue*
- *reindex*
- *runas*
- *scrape*
- *service*
- *sleep*
- *spin*
- *stats*
- *tag*
- *tee*
- *tree*
- *trigger*
- *uniq*
- *uptime*
- *vault*
- *version*
- *view*
- *wget*

See *Storm Reference - Document Syntax Conventions* for an explanation of the syntax format used below.

The Storm query language is covered in detail starting with the *Storm Reference - Introduction* section of the Synapse User Guide.

Tip: Storm commands, including custom commands, are added to Synapse as **runtime nodes** ("runt nodes" - see *Node, Runt*) of the form `syn:cmd`. With a few restrictions, these runt nodes can be lifted, filtered, and operated on similar to the way you work with other nodes.

Example

Lift the `syn:cmd` node for the Storm `movetag` command:

```
storm> syn:cmd=movetag
syn:cmd=movetag
        :doc = Rename an entire tag tree and preserve time intervals.
```

help

The `help` command displays the list of available commands within the current instance of Synapse and a brief message describing each command. Help for individual commands is available via `<command> --help`. The `help` command can also be used to inspect information about *Storm Libraries* and *Storm Types*.

Syntax:

```
storm> help --help


    List available information about Storm and brief descriptions of different items.

    Notes:

        If an item is provided, this can be a string or a function.

    Examples:

        // Get all available commands, libraries, types, and their brief descriptions.

        help

        // Only get commands which have "model" in the name.

        help model

        // Get help about the base Storm library

        help $lib

        // Get detailed help about a specific library or library function

        help --verbose $lib.print

        // Get detailed help about a named Storm type

        help --verbose str

        // Get help about a method from a $node object
```

(continues on next page)

(continued from previous page)

```
        <inbound $node> help $node.tags


Usage: help [options] <item>

Options:

  --help                      : Display the command usage.
  -v                          : Display detailed help when available.

Arguments:

  [item]                      : List information about a subset of commands or a␣
↪specific item.
```

aha

Storm includes `aha.*` commands that allow you to work with Synapse's AHA service, specifically with AHA service pools.

- *aha.pool.add*
- *aha.pool.del*
- *aha.pool.list*
- *aha.pool.svc.add*
- *aha.pool.svc.del*

Help for individual `auth.*` commands can be displayed using:

```
<command> --help
```

aha.pool.add

The `storm.aha.pool.add` command creates a new AHA service pool.

Syntax:

```
storm> aha.pool.add --help

Create an AHA service pool configuration.

Usage: aha.pool.add [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:
```

(continues on next page)

(continued from previous page)

```
  <name>                        : The name of the new AHA service pool.
```

aha.pool.del

The `storm.aha.pool.del` command deletes an AHA service pool configuration.

Syntax:

```
storm> aha.pool.del --help

Delete an AHA service pool configuration.

Usage: aha.pool.del [options] <name>

Options:

  --help                        : Display the command usage.

Arguments:

  <name>                        : The name of the AHA pool to delete.
```

aha.pool.list

The `storm.aha.pool.list` command lists AHA service pools and their associated services.

Syntax:

```
storm> aha.pool.list --help

Display a list of AHA service pools and their services.

Usage: aha.pool.list [options]

Options:

  --help                        : Display the command usage.
```

aha.pool.svc.add

The `storm.aha.pool.svc.add` command adds a service to an existing AHA service pool.

Syntax:

```
storm> aha.pool.svc.add --help


            Add an AHA service to a service pool.
```

(continues on next page)

(continued from previous page)

```
            Examples:

                // add 00.cortex... to the existing pool named pool.cortex
                aha.pool.svc.add pool.cortex... 00.cortex...


Usage: aha.pool.svc.add [options] <poolname> <svcname>

Options:

  --help                      : Display the command usage.

Arguments:

  <poolname>                  : The name of the AHA pool.
  <svcname>                   : The name of the AHA service.
```

aha.pool.svc.del

The `storm.aha.pool.svc.del` command removes a service from an existing AHA service pool.

Syntax:

```
storm> aha.pool.svc.del --help

Remove an AHA service from a service pool.

Usage: aha.pool.svc.del [options] <poolname> <svcname>

Options:

  --help                      : Display the command usage.

Arguments:

  <poolname>                  : The name of the AHA pool.
  <svcname>                   : The name of the AHA service.
```

auth

Storm includes `auth.*` commands that allow you create and manage users and roles, and manage their associated permissions (rules).

- *auth.gate.show*
- *auth.perms.list*
- *auth.role.add*
- *auth.role.addrule*
- *auth.role.del*
- *auth.role.delrule*

- *auth.role.list*
- *auth.role.mod*
- *auth.role.show*
- *auth.user.add*
- *auth.user.addrule*
- *auth.user.allowed*
- *auth.user.delrule*
- *auth.user.grant*
- *auth.user.list*
- *auth.user.mod*
- *auth.user.revoke*
- *auth.user.show*

Help for individual `auth.*` commands can be displayed using:

```
<command> --help
```

auth.gate.show

The `auth.gate.show` command displays the user, roles, and permissions associated with the specified *Auth Gate*.

Syntax

```
storm> auth.gate.show --help



            Display users, roles, and permissions for an auth gate.

            Examples:
                // Display the users and roles with permissions to the top layer of the␣
↪current view.
                auth.gate.show $lib.layer.get().iden

                // Display the users and roles with permissions to the current view.
                auth.gate.show $lib.view.get().iden


Usage: auth.gate.show [options] <gateiden>

Options:

  --help                      : Display the command usage.

Arguments:

  <gateiden>                  : The GUID of the auth gate.
```

auth.perms.list

The `auth.perms.list` command displays the set of permissions currently defined within the Cortex. This includes native Synapse permissions as well as any permissions associated with other packages and services, including Power-Ups. Each permission includes a brief description of the permission, the associated auth gate (e.g., 'cortex', 'layer') and the default state (true/allowed or false/denied).

Syntax:

```
storm> auth.perms.list --help

Display a list of the current permissions defined within the Cortex.

Usage: auth.perms.list [options]

Options:

  --help                      : Display the command usage.
```

auth.role.add

The `auth.role.add` command creates a role.

Syntax:

```
storm> auth.role.add --help


            Add a role.

            Examples:

                // Add a role named "ninjas"
                auth.role.add ninjas


Usage: auth.role.add [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the role.
```

auth.role.addrule

The `auth.role.addrule` command adds a rule (permission) to a role.

Syntax:

```
storm> auth.role.addrule --help


            Add a rule to a role.

            Examples:

                // add an allow rule to the role "ninjas" for permission "foo.bar.baz"
                auth.role.addrule ninjas foo.bar.baz

                // add a deny rule to the role "ninjas" for permission "foo.bar.baz"
                auth.role.addrule ninjas "!foo.bar.baz"

                // add an allow rule to the role "ninjas" for permission "baz" at the␣
↪first index.
                auth.role.addrule ninjas baz --index 0


Usage: auth.role.addrule [options] <name> <rule>

Options:

  --help                      : Display the command usage.
  --gate <gate>               : The auth gate id to add the rule to. (default: None)
  --index <index>             : Specify the rule location as a 0 based index. (default:␣
↪None)

Arguments:

  <name>                      : The name of the role.
  <rule>                      : The rule string.
```

auth.role.del

The `auth.role.del` command deletes a role.

Syntax:

```
storm> auth.role.del --help


            Delete a role.

            Examples:

                // Delete a role named "ninjas"
                auth.role.del ninjas
```

(continues on next page)

(continued from previous page)

```
Usage: auth.role.del [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the role.
```

auth.role.delrule

The `auth.role.delrule` command removes a rule (permission) from a role.

Syntax:

```
storm> auth.role.delrule --help


            Remove a rule from a role.

            Examples:

                // Delete the allow rule from the role "ninjas" for permission "foo.bar.
↪baz"
                auth.role.delrule ninjas foo.bar.baz

                // Delete the deny rule from the role "ninjas" for permission "foo.bar.
↪baz"
                auth.role.delrule ninjas "!foo.bar.baz"

                // Delete the rule at index 5 from the role "ninjas"
                auth.role.delrule ninjas --index  5


Usage: auth.role.delrule [options] <name> <rule>

Options:

  --help                      : Display the command usage.
  --gate <gate>               : The auth gate id to remove the rule from. (default: None)
  --index                     : Specify the rule as a 0 based index into the list of␣
↪rules.

Arguments:

  <name>                      : The name of the role.
  <rule>                      : The rule string.
```

auth.role.list

The `auth.role.list` lists all roles in the Cortex.

Syntax:

```
storm> auth.role.list --help


            List all roles.

            Examples:

                // Display the list of all roles
                auth.role.list


Usage: auth.role.list [options]

Options:

  --help                      : Display the command usage.
```

auth.role.mod

The `auth.role.mod` modifies an existing role.

Syntax:

```
storm> auth.role.mod --help


            Modify properties of a role.

            Examples:

                // Rename the "ninjas" role to "admins"
                auth.role.mod ninjas --name admins


Usage: auth.role.mod [options] <rolename>

Options:

  --help                      : Display the command usage.
  --name <name>               : The new name for the role.

Arguments:

  <rolename>                  : The name of the role.
```

auth.role.show

The `auth.role.show` displays the details for a given role.

Syntax:

```
storm> auth.role.show --help



            Display details for a given role by name.

            Examples:

                // Display details about the role "ninjas"
                auth.role.show ninjas


Usage: auth.role.show [options] <rolename>

Options:

  --help                      : Display the command usage.

Arguments:

  <rolename>                  : The name of the role.
```

auth.user.add

The `auth.user.add` command creates a user.

Syntax:

```
storm> auth.user.add --help


            Add a user.

            Examples:

                // Add a user named "visi" with the email address "visi@vertex.link"
                auth.user.add visi --email visi@vertex.link


Usage: auth.user.add [options] <name>

Options:

  --help                      : Display the command usage.
  --email <email>             : The user's email address. (default: None)

Arguments:
```

(continues on next page)

(continued from previous page)

```
  <name>                      : The name of the user.
```

auth.user.addrule

The `auth.user.addrule` command adds a rule (permission) to a user.

Syntax:

```
storm> auth.user.addrule --help


            Add a rule to a user.

            Examples:

                // add an allow rule to the user "visi" for permission "foo.bar.baz"
                auth.user.addrule visi foo.bar.baz

                // add a deny rule to the user "visi" for permission "foo.bar.baz"
                auth.user.addrule visi "!foo.bar.baz"

                // add an allow rule to the user "visi" for permission "baz" at the
↪first index.
                auth.user.addrule visi baz --index 0


Usage: auth.user.addrule [options] <name> <rule>

Options:

  --help                      : Display the command usage.
  --gate <gate>               : The auth gate id to grant permission on. (default: None)
  --index <index>             : Specify the rule location as a 0 based index. (default:
↪None)

Arguments:

  <name>                      : The name of the user.
  <rule>                      : The rule string.
```

auth.user.allowed

The `auth.user.allowed` command checks whether a user has a permission for the specified scope (view or layer; if no scope is specified with the `--gate` option, the permission is checked globally).

The command retuns whether the permission is allowed (true) the source of the permission (e.g., if the permission is due to having a particular role).

Syntax:

```
storm> auth.user.allowed --help


            Show whether the user is allowed the given permission and why.

            Examples:

                auth.user.allowed visi foo.bar


Usage: auth.user.allowed [options] <username> <permname>

Options:

  --help                      : Display the command usage.
  --gate <gate>               : An auth gate to test the perms against.

Arguments:

  <username>                  : The name of the user.
  <permname>                  : The permission string.
```

auth.user.delrule

The `auth.user.delrule` command removes a rule (permission) from a user.

Syntax:

```
storm> auth.user.delrule --help


            Remove a rule from a user.

            Examples:

                // Delete the allow rule from the user "visi" for permission "foo.bar.baz
↪"
                auth.user.delrule visi foo.bar.baz

                // Delete the deny rule from the user "visi" for permission "foo.bar.baz"
                auth.user.delrule visi "!foo.bar.baz"

                // Delete the rule at index 5 from the user "visi"
                auth.user.delrule visi --index  5


Usage: auth.user.delrule [options] <name> <rule>

Options:

  --help                      : Display the command usage.
  --gate <gate>               : The auth gate id to grant permission on. (default: None)
```

(continues on next page)

(continued from previous page)

```
  --index                     : Specify the rule as a 0 based index into the list of␣
↪rules.

Arguments:

  <name>                      : The name of the user.
  <rule>                      : The rule string.
```

auth.user.grant

The `auth.user.grant` command grants a role (and its associated permissions) to a user.

Syntax:

```
storm> auth.user.grant --help


            Grant a role to a user.

            Examples:

                // Grant the role "ninjas" to the user "visi"
                auth.user.grant visi ninjas

                // Grant the role "ninjas" to the user "visi" at the first index.
                auth.user.grant visi ninjas --index 0



Usage: auth.user.grant [options] <username> <rolename>

Options:

  --help                      : Display the command usage.
  --index <index>             : Specify the role location as a 0 based index. (default:␣
↪None)

Arguments:

  <username>                  : The name of the user.
  <rolename>                  : The name of the role.
```

auth.user.list

The `auth.user.list` command displays all users in the Cortex.

Syntax:

```
storm> auth.user.list --help


            List all users.

            Examples:

                // Display the list of all users
                auth.user.list


Usage: auth.user.list [options]

Options:

  --help                      : Display the command usage.
```

auth.user.mod

The `auth.user.mod` command modifies a user account.

Syntax:

```
storm> auth.user.mod --help


            Modify properties of a user.

            Examples:

                // Rename the user "foo" to "bar"
                auth.user.mod foo --name bar

                // Make the user "visi" an admin
                auth.user.mod visi --admin $lib.true

                // Unlock the user "visi" and set their email to "visi@vertex.link"
                auth.user.mod visi --locked $lib.false --email visi@vertex.link

                // Grant admin access to user visi for the current view
                auth.user.mod visi --admin $lib.true --gate $lib.view.get().iden

                // Revoke admin access to user visi for the current view
                auth.user.mod visi --admin $lib.false --gate $lib.view.get().iden


Usage: auth.user.mod [options] <username>
```

(continues on next page)

(continued from previous page)

```
Options:

  --help                        : Display the command usage.
  --name <name>                 : The new name for the user.
  --email <email>               : The email address to set for the user.
  --passwd <passwd>             : The new password for the user. This is best passed into␣
↪the runtime as a variable.
  --admin <admin>               : True to make the user and admin, false to remove their␣
↪remove their admin status.
  --gate <gate>                 : The auth gate iden to grant or revoke admin status on.␣
↪Use in conjunction with `--admin <bool>`.
  --locked <locked>             : True to lock the user, false to unlock them.

Arguments:

  <username>                    : The name of the user.
```

auth.user.revoke

The `auth.user.revoke` command revokes a role (and its associated permissions) from a user.

Syntax:

```
storm> auth.user.revoke --help


            Revoke a role from a user.

            Examples:

                // Revoke the role "ninjas" from the user "visi"
                auth.user.revoke visi ninjas



Usage: auth.user.revoke [options] <username> <rolename>

Options:

  --help                        : Display the command usage.

Arguments:

  <username>                    : The name of the user.
  <rolename>                    : The name of the role.
```

auth.user.show

The `auth.user.show` command displays information for a specific user.

Syntax:

```
storm> auth.user.show --help


            Display details for a given user by name.

            Examples:

                // Display details about the user "visi"
                auth.user.show visi


Usage: auth.user.show [options] <username>

Options:

  --help                      : Display the command usage.

Arguments:

  <username>                  : The name of the user.
```

background

The `background` command allows you to execute a Storm query as a background task (e.g., to free up the CLI / Storm runtime for additional queries).

Note: Use of `background` is a "fire-and-forget" process - any status messages (warnings or errors) are not returned to the console, and if the query is interrupted for any reason, it will not resume.

See also *parallel*.

Syntax:

```
storm> background --help


    Execute a query pipeline as a background task.
    NOTE: Variables are passed through but nodes are not


Usage: background [options] <query>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  <query>                     : The query to execute in the background.
```

batch

The batch command allows you to run a Storm query with batched sets of nodes.

Note that in most cases, Storm queries are meant to operate in a "streaming" manner on individual nodes. This command is intended to be used in cases such as querying external APIs that support aggregate queries (i.e., an API that allows you to query 100 objects in a single API call as part of the API's quota system).

Syntax:

```
storm> batch --help


    Run a query with batched sets of nodes.

    The batched query will have the set of inbound nodes available in the
    variable $nodes.

    This command also takes a conditional as an argument. If the conditional
    evaluates to true, the nodes returned by the batched query will be yielded,
    if it evaluates to false, the inbound nodes will be yielded after executing the
    batched query.

    NOTE: This command is intended to facilitate use cases such as queries to external
          APIs with aggregate node values to reduce quota consumption. As this command
          interrupts the node stream, it should be used carefully to avoid unintended
          slowdowns in the pipeline.

    Example:

        // Execute a query with batches of 5 nodes, then yield the inbound nodes
        batch $lib.false --size 5 { $lib.print($nodes) }


Usage: batch [options] <cond> <query>

Options:

  --help                      : Display the command usage.
  --size <size>               : The number of nodes to collect before running the␣
↪batched query (max 10000). (default: 10)

Arguments:

  <cond>                      : The conditional value for the yield option.
  <query>                     : The query to execute with batched nodes.
```

copyto

The copyto command allows you to copy nodes from the current view to a specified target view. Nodes are copied to the write layer (the topmost layer) in the target view.

When copying nodes, the history of the node (i.e., changes to the node, timestamps, associated user) in the **source** view (the view layer(s)) is preserved; the changes written to the **target** view's write layer are owned by the user executing the copyto command.

See the *movenodes* command to move nodes between layers in the same layer stack.

Note: The copyto command, like the movenodes command, is meant to be used by Synapse **administrators** in specific use cases.

Syntax:

```
storm> copyto --help


    Copy nodes from the current view into another view.

    Examples:

        // Copy all nodes tagged with #cno.mal.redtree to the target view.

        #cno.mal.redtree | copyto 33c971ac77943da91392dadd0eec0571


Usage: copyto [options] <view>

Options:

  --help                      : Display the command usage.
  --no-data                   : Do not copy node data to the destination view.

Arguments:

  <view>                      : The destination view ID to copy the nodes to.
```

cortex.httpapi

Note: See the *Extended HTTP API* guide for additional background on Extended HTTP API endpoints.

Storm includes cortex.httpapi.* commands that allow a user to list and manage Extended HTTP API endpoints.

- *cortex.httpapi.index*
- *cortex.httpapi.list*
- *cortex.httpapi.stat*

Help for individual cortex.httpapi.* commands can be displayed using:

```
<command> --help
```

cortex.httpapi.index

The `cortex.httpapi.index` command is used to change the resolution order of the Extended HTTP API endpoints.

Syntax:

```
storm> cortex.httpapi.index --help

Set the index of an Extended HTTP API endpoint.

Examples:

    // Move an endpoint to the first index.
    cortex.httpapi.index 60e5ba38e90958fd8e2ddd9e4730f16b 0

    // Move an endpoint to the third index.
    cortex.httpapi.index dd9e4730f16b60e5ba58fd8e2d38e909 2


Usage: cortex.httpapi.index [options] <iden> <index>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : The iden of the endpoint to move. This will also match␣
↪iden prefixes or name prefixes.
  <index>                     : Specify the endpoint location as a 0 based index.
```

cortex.httpapi.list

The `cortex.httpapi.list` command is used to list the Extended HTTP API endpoints.

Syntax:

```
storm> cortex.httpapi.list --help

List Extended HTTP API endpoints

Usage: cortex.httpapi.list [options]

Options:

  --help                      : Display the command usage.
```

cortex.httpapi.stat

The `cortex.httpapi.stat` command is used to show the detailed information for a single Extended HTTP API Endpoint.

Syntax:

```
storm> cortex.httpapi.stat --help

Get details for an Extended HTTP API endpoint.

Usage: cortex.httpapi.stat [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : The iden of the endpoint to inspect. This will also␣
↪match iden prefixes or name prefixes.
```

count

The `count` command enumerates the number of nodes returned from a given Storm query and displays the final tally. The associated nodes can optionally be displayed with the `--yield` switch.

Syntax:

```
storm> count --help


    Iterate through query results, and print the resulting number of nodes
    which were lifted. This does not yield the nodes counted, unless the
    --yield switch is provided.

    Example:

        # Count the number of IPV4 nodes with a given ASN.
        inet:ipv4:asn=20 | count

        # Count the number of IPV4 nodes with a given ASN and yield them.
        inet:ipv4:asn=20 | count --yield



Usage: count [options]

Options:

  --help                      : Display the command usage.
  --yield                     : Yield inbound nodes.
```

Examples:

- Count the number of IP address nodes that Trend Micro reports are associated with the threat group Earth Preta:

```
storm> inet:ipv4#rep.trend.earthpreta | count
Counted 5 nodes.
```

- Count nodes from a lift and yield the output:

```
storm> inet:ipv4#rep.trend.earthpreta | count --yield
inet:ipv4=66.129.222.1
        :type = unicast
        .created = 2024/04/19 14:26:23.645
        #rep.trend.earthpreta
inet:ipv4=184.82.164.104
        :type = unicast
        .created = 2024/04/19 14:26:23.652
        #rep.trend.earthpreta
inet:ipv4=209.161.249.125
        :type = unicast
        .created = 2024/04/19 14:26:23.658
        #rep.trend.earthpreta
inet:ipv4=69.90.65.240
        :type = unicast
        .created = 2024/04/19 14:26:23.664
        #rep.trend.earthpreta
inet:ipv4=70.62.232.98
        :type = unicast
        .created = 2024/04/19 14:26:23.669
        #rep.trend.earthpreta
Counted 5 nodes.
```

- Count the number of DNS A records for the domain woot.com where the lift produces no results:

```
storm> inet:dns:a:fqdn=woot.com | count
Counted 0 nodes.
```

cron

Note: See the *Storm Reference - Automation* guide for additional background on cron jobs (as well as triggers and macros), including examples.

Storm includes `cron.*` commands that allow you to create and manage scheduled *Cron* jobs. Within Synapse, jobs are Storm queries that execute on a recurring or one-time (`cron.at`) basis.

- *cron.add*
- *cron.at*
- *cron.cleanup*
- *cron.list*
- *cron.stat*
- *cron.mod*

- *cron.move*
- *cron.disable*
- *cron.enable*
- *cron.del*

Help for individual `cron.*` commands can be displayed using:

```
<command> --help
```

Tip: Cron jobs (including jobs created with `cron.at`) are added to Synapse as **runtime nodes** ("runt nodes" - see *Node, Runt*) of the form `syn:cron`. With a few restrictions, these runt nodes can be lifted, filtered, and operated on similar to the way you work with other nodes.

cron.add

The `cron.add` command creates an individual cron job within a Cortex.

Syntax:

```
storm> cron.add --help


Add a recurring cron job to a cortex.

Notes:
    All times are interpreted as UTC.

    All arguments are interpreted as the job period, unless the value ends in
    an equals sign, in which case the argument is interpreted as the recurrence
    period.  Only one recurrence period parameter may be specified.

    Currently, a fixed unit must not be larger than a specified recurrence
    period.  i.e. '--hour 7 --minute +15' (every 15 minutes from 7-8am?) is not
    supported.

    Value values for fixed hours are 0-23 on a 24-hour clock where midnight is 0.

    If the --day parameter value does not start with a '+' and is an integer, it is
    interpreted as a fixed day of the month.  A negative integer may be
    specified to count from the end of the month with -1 meaning the last day
    of the month.  All fixed day values are clamped to valid days, so for
    example '-d 31' will run on February 28.
    If the fixed day parameter is a value in ([Mon, Tue, Wed, Thu, Fri, Sat,
    Sun] if locale is set to English) it is interpreted as a fixed day of the
    week.

    Otherwise, if the parameter value starts with a '+', then it is interpreted
    as a recurrence interval of that many days.

    If no plus-sign-starting parameter is specified, the recurrence period
    defaults to the unit larger than all the fixed parameters.   e.g. '--minute 5'
```

(continues on next page)

(continued from previous page)

```
    means every hour at 5 minutes past, and --hour 3, --minute 1 means 3:01 every day.

    At least one optional parameter must be provided.

    All parameters accept multiple comma-separated values.  If multiple
    parameters have multiple values, all combinations of those values are used.

    All fixed units not specified lower than the recurrence period default to
    the lowest valid value, e.g. --month +2 will be scheduled at 12:00am the first of
    every other month.  One exception is if the largest fixed value is day of the
    week, then the default period is set to be a week.

    A month period with a day of week fixed value is not currently supported.

    Fixed-value year (i.e. --year 2019) is not supported.  See the 'at'
    command for one-time cron jobs.

    As an alternative to the above options, one may use exactly one of
    --hourly, --daily, --monthly, --yearly with a colon-separated list of
    fixed parameters for the value.  It is an error to use both the individual
    options and these aliases at the same time.

Examples:
    Run a query every last day of the month at 3 am
    cron.add --hour 3 --day -1 {#foo}

    Run a query every 8 hours
    cron.add --hour +8 {#foo}

    Run a query every Wednesday and Sunday at midnight and noon
    cron.add --hour 0,12 --day Wed,Sun {#foo}

    Run a query every other day at 3:57pm
    cron.add --day +2 --minute 57 --hour 15 {#foo}


Usage: cron.add [options] <query>

Options:

  --help                      : Display the command usage.
  --pool                      : Allow the cron job to be run by a mirror from the query␣
↪pool.
  --minute <minute>           : Minute value for job or recurrence period.
  --name <name>               : An optional name for the cron job.
  --doc <doc>                 : An optional doc string for the cron job.
  --hour <hour>               : Hour value for job or recurrence period.
  --day <day>                 : Day value for job or recurrence period.
  --month <month>             : Month value for job or recurrence period.
  --year <year>               : Year value for recurrence period.
  --hourly <hourly>           : Fixed parameters for an hourly job.
  --daily <daily>             : Fixed parameters for a daily job.
```

(continues on next page)

(continued from previous page)

```
  --monthly <monthly>         : Fixed parameters for a monthly job.
  --yearly <yearly>           : Fixed parameters for a yearly job.
  --iden <iden>               : Fixed iden to assign to the cron job
  --view <view>               : View to run the cron job against

Arguments:

  <query>                     : Query for the cron job to execute.
```

cron.at

The `cron.at` command creates a non-recurring (one-time) cron job within a Cortex. Just like standard (recurring) cron jobs, jobs created with `cron.at` will persist (remain in the list of cron jobs and as `syn:cron` runt nodes) until they are explicitly removed using `cron.del`.

Syntax:

```
storm> cron.at --help


Adds a non-recurring cron job.

Notes:
    This command accepts one or more time specifications followed by exactly
    one storm query in curly braces.  Each time specification may be in synapse
    time delta format (e.g --day +1) or synapse time format (e.g.
    20501217030432101).  Seconds will be ignored, as cron jobs' granularity is
    limited to minutes.

    All times are interpreted as UTC.

    The other option for time specification is a relative time from now.  This
    consists of a plus sign, a positive integer, then one of 'minutes, hours,
    days'.

    Note that the record for a cron job is stored until explicitly deleted via
    "cron.del".

Examples:
    # Run a storm query in 5 minutes
    cron.at --minute +5 {[inet:ipv4=1]}

    # Run a storm query tomorrow and in a week
    cron.at --day +1,+7 {[inet:ipv4=1]}

    # Run a query at the end of the year Zulu
    cron.at --dt 20181231Z2359 {[inet:ipv4=1]}


Usage: cron.at [options] <query>
```

(continues on next page)

(continued from previous page)

```
Options:

  --help                      : Display the command usage.
  --minute <minute>           : Minute(s) to execute at.
  --hour <hour>               : Hour(s) to execute at.
  --day <day>                 : Day(s) to execute at.
  --dt <dt>                   : Datetime(s) to execute at.
  --now                       : Execute immediately.
  --iden <iden>               : A set iden to assign to the new cron job
  --view <view>               : View to run the cron job against

Arguments:

  <query>                     : Query for the cron job to execute.
```

cron.cleanup

The `cron.cleanup` command can be used to remove any one-time cron jobs ("at" jobs) that have completed.

Syntax:

```
storm> cron.cleanup --help

Delete all completed at jobs

Usage: cron.cleanup [options]

Options:

  --help                      : Display the command usage.
```

cron.list

The `cron.list` command displays the set of cron jobs in the Cortex that the current user can view / modify based on their permissions.

Cron jobs are displayed in alphanumeric order by job *Iden*. Jobs are sorted upon Cortex initialization, so newly-created jobs will be displayed at the bottom of the list until the list is re-sorted the next time the Cortex is restarted.

Syntax:

```
storm> cron.list --help

List existing cron jobs in the cortex.

Usage: cron.list [options]

Options:

  --help                      : Display the command usage.
```

cron.stat

The `cron.stat` command displays statistics for an individual cron job and provides more detail on an individual job vs. `cron.list`, including any errors and the interval at which the job executes. To view the stats for a job, you must provide the first portion of the job's iden (i.e., enough of the iden that the job can be uniquely identified), which can be obtained using `cron.list` or by lifting the appropriate `syn:cron` node.

Syntax:

```
storm> cron.stat --help

Gives detailed information about a cron job.

Usage: cron.stat [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
```

cron.mod

The `cron.mod` command modifies the Storm query associated with a specific cron job. To modify a job, you must provide the first portion of the job's iden (i.e., enough of the iden that the job can be uniquely identified), which can be obtained using `cron.list` or by lifting the appropriate `syn:cron` node.

Note: Other aspects of the cron job, such as its schedule for execution, cannot be modified once the job has been created. To change these aspects you must delete and re-add the job.

Syntax:

```
storm> cron.mod --help

Modify an existing cron job's query.

Usage: cron.mod [options] <iden> <query>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
  <query>                     : New storm query for the cron job.
```

cron.move

The `cron.move` command moves a cron job from one *View* to another.

Syntax:

```
storm> cron.move --help

Move a cron job from one view to another

Usage: cron.move [options] <iden> <view>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
  <view>                      : View to move the cron job to.
```

cron.disable

The `cron.disable` command disables a job and prevents it from executing without removing it from the Cortex. To disable a job, you must provide the first portion of the job's iden (i.e., enough of the iden that the job can be uniquely identified), which can be obtained using `cron.list` or by lifting the appropriate `syn:cron` node.

Syntax:

```
storm> cron.disable --help

Disable a cron job in the cortex.

Usage: cron.disable [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
```

cron.enable

The `cron.enable` command enables a disabled cron job. To enable a job, you must provide the first portion of the job's iden (i.e., enough of the iden that the job can be uniquely identified), which can be obtained using `cron.list` or by lifting the appropriate `syn:cron` node.

Note: Cron jobs, including non-recurring jobs added with `cron.at`, are enabled by default upon creation.

Syntax:

```
storm> cron.enable --help

Enable a cron job in the cortex.

Usage: cron.enable [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
```

cron.del

The `cron.del` command permanently removes a cron job from the Cortex. To delete a job, you must provide the first portion of the job's iden (i.e., enough of the iden that the job can be uniquely identified), which can be obtained using `cron.list` or by lifting the appropriate `syn:cron` node.

Syntax:

```
storm> cron.del --help

Delete a cron job from the cortex.

Usage: cron.del [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid cron job iden␣
↪is accepted.
```

delnode

The `delnode` command deletes a node or set of nodes from a Cortex.

> **Warning:** The Storm `delnode` command includes some limited checks (see below) to try and prevent the accidental deletion of nodes that are still connected to other nodes in the knowledge graph. However, these checks are not foolproof, and `delnode` has the potential to be destructive if executed on an incorrect, badly formed, or mistyped query.
>
> Users are **strongly encouraged** to validate their query by first executing it on its own to confirm it returns the expected nodes before piping the query to the `delnode` command.
>
> In addition, use of the `--force` switch with `delnode` will override all safety checks and forcibly delete ALL nodes input to the command.
>
> **This parameter should be used with extreme caution as it may result in broken references (e.g., "holes" in the graph) within Synapse.**

Syntax:

```
storm> delnode --help


    Delete nodes produced by the previous query logic.

    (no nodes are returned)

    Example

        inet:fqdn=vertex.link | delnode


Usage: delnode [options]

Options:

  --help                      : Display the command usage.
  --force                     : Force delete even if it causes broken references␣
↪(requires admin).
  --delbytes                  : For file:bytes nodes, remove the bytes associated with␣
↪the sha256 property from the axon as well if present.
  --deledges                  : Delete N2 light edges before deleting the node.
```

Examples:

- Delete the node for the domain woowoo.com:

```
storm> inet:fqdn=woowoo.com | delnode
```

- Forcibly delete all nodes with the #testing tag:

```
storm> #testing | delnode --force
```

Usage Notes:

- `delnode` operates on the output of a previous Storm query.

- delnode performs some basic sanity-checking to help prevent egregious mistakes, and will generate an error in cases such as:
  - attempting to delete a node (such as inet:fqdn=woot.com) that is still referenced by (i.e., is a secondary property of) another node (such as inet:dns:a=(woot.com, 1.1.1.1).
  - attmpting to delete a syn:tag node where that tag still exists on other nodes.

  However, it is important to keep in mind that **delnode cannot prevent all mistakes.**

diff

The diff command generates a list of nodes with changes (i.e., newly created or modified nodes) present in the top *Layer* of the current *View*. The diff command may be useful before performing a *merge* operation.

Syntax:

```
storm> diff --help


    Generate a list of nodes with changes in the top layer of the current view.

    Examples:

        // Lift all nodes with any changes

        diff

        // Lift ou:org nodes that were added in the top layer.

        diff --prop ou:org

        // Lift inet:ipv4 nodes with the :asn property modified in the top layer.

        diff --prop inet:ipv4:asn

        // Lift the nodes with the tag #cno.mal.redtree added in the top layer.

        diff --tag cno.mal.redtree


Usage: diff [options]

Options:

  --help                      : Display the command usage.
  --tag <tag>                 : Lift only nodes with the given tag in the top layer.␣
↪(default: None)
  --prop <prop>               : Lift nodes with changes to the given property the top␣
↪layer. (default: None)
```

divert

The divert command allows Storm to either consume a generator or yield its results based on a conditional.

Syntax:

```
storm> divert --help


    Either consume a generator or yield it's results based on a conditional.

    NOTE: This command is purpose built to facilitate the --yield convention
          common to storm commands.

    NOTE: The genr argument must not be a function that returns, else it will
          be invoked for each inbound node.

    Example:
        divert $cmdopts.yield $fooBarBaz()


Usage: divert [options] <cond> <genr>

Options:

  --help                      : Display the command usage.
  --size <size>               : The max number of times to iterate the generator.␣
↪(default: None)

Arguments:

  <cond>                      : The conditional value for the yield option.
  <genr>                      : The generator function value that yields nodes.
```

dmon

Storm includes dmon.* commands that allow you to work with daemons (see *Daemon*).

- *dmon.list*

Help for individual dmon.* commands can be displayed using:

```
<command> --help
```

dmon.list

The dmon.list command displays the set of running dmon queries in the Cortex.

Syntax:

```
storm> dmon.list --help

List the storm daemon queries running in the cortex.

```

(continues on next page)

(continued from previous page)

```
Usage: dmon.list [options]

Options:

  --help                      : Display the command usage.
```

edges

Storm includes `edges.*` commands that allow you to work with lightweight (light) edges. Also see the `lift.byverb` and `model.edge.*` commands under *lift* and *model* below.

- *edges.del*

Help for individual `edge.*` commands can be displayed using:

`<command> --help`

edges.del

The `edges.del` command is designed to delete multiple light edges to (or from) a set of nodes (contrast with using Storm edit syntax - see *Delete Light Edges*).

Syntax:

```
storm> edges.del --help


    Bulk delete light edges from input nodes.

    Examples:

        # Delete all "foo" light edges from an inet:ipv4
        inet:ipv4=1.2.3.4 | edges.del foo

        # Delete light edges with any verb from a node
        inet:ipv4=1.2.3.4 | edges.del *

        # Delete all "foo" light edges to an inet:ipv4
        inet:ipv4=1.2.3.4 | edges.del foo --n2


Usage: edges.del [options] <verb>

Options:

  --help                      : Display the command usage.
  --n2                        : Delete light edges where input node is N2 instead of N1.

Arguments:

  <verb>                      : The verb of light edges to delete.
```

feed

Storm includes `feed.*` commands that allow you to work with feeds (see *Feed*).

- *feed.list*

Help for individual `feed.*` commands can be displayed using:

```
<command> --help
```

feed.list

The `feed.list` command displays available feed functions in the Cortex.

Syntax:

```
storm> feed.list --help

List the feed functions available in the Cortex

Usage: feed.list [options]

Options:

  --help                      : Display the command usage.
```

gen

Storm includes various `gen.*` ("generate") commands that allow you to easily query for common guid-based nodes (see *Form*, *GUID*) based on one or more "human friendly" secondary properties, and create (generate) the specified node if it does not already exist.

Because guid nodes have a primary property that may be arbitrary, `gen.*` commands simplify the process of **deconflicting on secondary properties** before creating certain guid nodes.

Note: See the *guid* section of the *Storm Reference - Type-Specific Storm Behavior* for a detailed discussion of guids, guid behavior, and deconfliction considerations for guid forms.

Nodes created using generate commands will have a limited subset of properties set (e.g., an organization node deconflicted and created based on a name will only have its `ou:org:name` property set). Users can set additional property values as they see fit.

- *gen.it.prod.soft*
- *gen.lang.language*
- *gen.ou.campaign*
- *gen.ou.id.number*
- *gen.ou.id.type*
- *gen.ou.industry*
- *gen.ou.org*
- *gen.ou.org.hq*

- *gen.pol.country*
- *gen.pol.country.government*
- *gen.ps.contact.email*
- *gen.risk.threat*
- *gen.risk.tool.software*
- *gen.risk.vuln*

Help for individual `gen.*` commands can be displayed using:

```
<command> --help
```

Note: New `gen.*` commands are added to Synapse on an ongoing basis as we identify new cases where such commands are helpful. Use the `help` command for the current list of `gen.*` commands available in your instance of Synapse.

---

gen.it.prod.soft

The `gen.it.prod.soft` command locates (lifts) or creates an `it:prod:soft` node based on the software name (`it:prod:soft:name` and / or `it:prod:soft:names`).

Syntax:

```
storm> gen.it.prod.soft --help

Lift (or create) an it:prod:soft node based on the software name.

Usage: gen.it.prod.soft [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the software.
```

gen.lang.language

The `gen.lang.language` command locates (lifts) or creates a `lang:language` node based on the language name (`lang:language:name` and / or `lang:language:names`).

Syntax:

```
storm> gen.lang.language --help

Lift (or create) a lang:language node based on the name.

Usage: gen.lang.language [options] <name>
```

(continues on next page)

(continued from previous page)

```
Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the language.
```

gen.ou.campaign

The `gen.ou.campaign` command locates (lifts) or creates an `ou:campaign` node based on the campaign name (`ou:campaign:name` and / or `ou:campaign:names`) and the name of the reporting organization (`ou:campaign:reporter:name`).

Syntax:

```
storm> gen.ou.campaign --help

Lift (or create) an ou:campaign based on the name and reporting organization.

Usage: gen.ou.campaign [options] <name> <reporter>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the campaign.
  <reporter>                  : The name of the reporting organization.
```

gen.ou.id.number

The `gen.ou.id.number` command locates (lifts) or creates an `ou:id:number` node based on the organization ID type (`ou:id:type`) and organization ID value (`ou:id:value`).

Syntax:

```
storm> gen.ou.id.number --help

Lift (or create) an ou:id:number node based on the organization ID type and value.

Usage: gen.ou.id.number [options] <type> <value>

Options:

  --help                      : Display the command usage.

Arguments:

```

(continues on next page)

(continued from previous page)

```
  <type>                      : The type of the organization ID.
  <value>                     : The value of the organization ID.
```

gen.ou.id.type

The `gen.ou.id.type` command locates (lifts) or creates an `ou:id:type` node based on the friendly name of the organization ID type (`ou:id:type:name`).

Syntax:

```
storm> gen.ou.id.type --help

Lift (or create) an ou:id:type node based on the name of the type.

Usage: gen.ou.id.type [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The friendly name of the organization ID type.
```

gen.ou.industry

The `gen.ou.industry` commands locates (lifts) or creates an `ou:industry` node based on the industry name (`ou:industry:name` and / or `ou:industry:names`).

Syntax:

```
storm> gen.ou.industry --help


            Lift (or create) an ou:industry node based on the industry name.


Usage: gen.ou.industry [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The industry name.
```

gen.ou.org

The `gen.ou.org` command locates (lifts) or creates an `ou:org` node based on the organization name (`ou:org:name` and / or `ou:org:names`).

Syntax:

```
storm> gen.ou.org --help

Lift (or create) an ou:org node based on the organization name.

Usage: gen.ou.org [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the organization.
```

gen.ou.org.hq

The `gen.ou.org.hq` command locates (lifts) the primary `ps:contact` node for an organization (i.e., the contact set for the `ou:org:hq` property) or creates the contact node (and sets the `ou:org:hq` property) if it does not exist, based on the organization name (`ou:org:name` and / or `ou:org:names`).

Syntax:

```
storm> gen.ou.org.hq --help

Lift (or create) the primary ps:contact node for the ou:org based on the organization␣
↪name.

Usage: gen.ou.org.hq [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the organization.
```

gen.pol.country

The `gen.pol.country` command locates (lifts) or creates a `pol:country` node based on the two-letter ISO-3166 country code (`pol:country:iso2`) .

Syntax:

```
storm> gen.pol.country --help


            Lift (or create) a pol:country node based on the 2 letter ISO-3166 country␣
↪code.

            Examples:

                // Yield the pol:country node which represents the country of Ukraine.
                gen.pol.country ua


Usage: gen.pol.country [options] <iso2>

Options:

  --help                      : Display the command usage.
  --try                       : Type normalization will fail silently instead of raising␣
↪an exception.

Arguments:

  <iso2>                      : The 2 letter ISO-3166 country code.
```

gen.pol.country.government

The `gen.pol.country.government` command locates (lifts) the `ou:org` node representing a country's government (i.e., the organization set for the `pol:country:government` property) or creates the node (and sets the `pol:country:government` property) if it does not exist, based on the two-letter ISO-3166 country code (`pol:country:iso2`).

Syntax:

```
storm> gen.pol.country.government --help


            Lift (or create) the ou:org node representing a country's government based␣
↪on the 2 letter ISO-3166 country code.

            Examples:

                // Yield the ou:org node which represents the Government of Ukraine.
                gen.pol.country.government ua


Usage: gen.pol.country.government [options] <iso2>
```

(continues on next page)

(continued from previous page)

```
Options:

  --help                      : Display the command usage.
  --try                       : Type normalization will fail silently instead of raising␣
↪an exception.

Arguments:

  <iso2>                      : The 2 letter ISO-3166 country code.
```

gen.ps.contact.email

The `gen.ps.contact.email` command locates (lifts) or creates a `ps:contact` node using the contact's primary email address (`ps:contact:email`) and type (`ps:contact:type`).

Syntax:

```
storm> gen.ps.contact.email --help


            Lift (or create) the ps:contact node by deconflicting the email and type.

            Examples:

                // Yield the ps:contact node for the type and email
                gen.ps.contact.email vertex.employee visi@vertex.link


Usage: gen.ps.contact.email [options] <type> <email>

Options:

  --help                      : Display the command usage.
  --try                       : Type normalization will fail silently instead of raising␣
↪an exception.

Arguments:

  <type>                      : The contact type.
  <email>                     : The contact email address.
```

gen.risk.threat

The `gen.risk.threat` command locates (lifts) or creates a `risk:threat` node using the name of the threat group (`risk:threat:org:name` and / or `risk:threat:org:names`) and the name of the entity reporting on the threat (`risk:threat:reporter:name`).

Syntax:

```
storm> gen.risk.threat --help


            Lift (or create) a risk:threat node based on the threat name and reporter␣
↪name.

            Examples:

                // Yield a risk:threat node for the threat cluster "APT1" reported by
↪"Mandiant".
                gen.risk.threat apt1 mandiant


Usage: gen.risk.threat [options] <name> <reporter>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the threat cluster. For example: APT1
  <reporter>                  : The name of the reporting organization. For example:␣
↪Mandiant
```

gen.risk.tool.software

The `gen.risk.tool.software` command locates (lifts) or creates a `risk:tool:software` node using the name of the software / malware (`risk:tool:software:soft:name` and / or `risk:software:soft:names`) and the name of the entity reporting on the software / malware (`risk:tool:software:reporter:name`).

Syntax:

```
storm> gen.risk.tool.software --help


            Lift (or create) a risk:tool:software node based on the tool name and␣
↪reporter name.

            Examples:

                // Yield a risk:tool:software node for the "redtree" tool reported by
↪"vertex".
                gen.risk.tool.software redtree vertex

```

(continues on next page)

(continued from previous page)

```
Usage: gen.risk.tool.software [options] <name> <reporter>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The tool name.
  <reporter>                  : The name of the reporting organization. For example:
↪"recorded future"
```

gen.risk.vuln

The `gen.risk.vuln` command locates (lifts) or creates a `risk:tool:vuln` node using the Common Vulnerabilities and Exposures (CVE) number associated with the vulnerability (`risk:vuln:cve`).

Syntax:

```
storm> gen.risk.vuln --help


            Lift (or create) a risk:vuln node based on the CVE and reporter name.

            Examples:

                // Yield a risk:vuln node for CVE-2012-0157 reported by Mandiant.
                gen.risk.vuln CVE-2012-0157 Mandiant


Usage: gen.risk.vuln [options] <cve> <reporter>

Options:

  --help                      : Display the command usage.
  --try                       : Type normalization will fail silently instead of raising␣
↪an exception.

Arguments:

  <cve>                       : The CVE identifier.
  [reporter]                  : The name of the reporting organization.
```

graph

The graph command generates a subgraph based on a specified set of nodes and parameters.

Syntax:

```
storm> graph --help


    Generate a subgraph from the given input nodes and command line options.

    Example:

        Using the graph command::

            inet:fqdn | graph
                        --degrees 2
                        --filter { -#nope }
                        --pivot { -> meta:seen }
                        --form-pivot inet:fqdn {<- * | limit 20}
                        --form-pivot inet:fqdn {-> * | limit 20}
                        --form-filter inet:fqdn {-inet:fqdn:issuffix=1}
                        --form-pivot syn:tag {-> *}
                        --form-pivot * {-> #}



Usage: graph [options]

Options:

  --help                        : Display the command usage.
  --degrees <degrees>           : How many degrees to graph out. (default: 1)
  --pivot <pivot>               : Specify a storm pivot for all nodes. (must quote)␣
↪(default: [])
  --filter <filter>             : Specify a storm filter for all nodes. (must quote)␣
↪(default: [])
  --no-edges                    : Do not include light weight edges in the per-node output.
  --form-pivot <form_pivot>     : Specify a <form> <pivot> form specific pivot. (default:␣
↪[])
  --form-filter <form_filter> : Specify a <form> <filter> form specific filter.␣
↪(default: [])
  --refs                        : Do automatic in-model pivoting with node.getNodeRefs().
  --yield-filtered              : Yield nodes which would be filtered. This still performs␣
↪pivots to collect edge data,but does not yield pivoted nodes.
  --no-filter-input             : Do not drop input nodes if they would match a filter.
```

iden

The `iden` command lifts one or more nodes by their node identifier (node ID / iden).

Syntax:

```
storm> iden --help


    Lift nodes by iden.

    Example:

        iden b25bc9eec7e159dce879f9ec85fb791f83b505ac55b346fcb64c3c51e98d1175 | count


Usage: iden [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  [<iden> ...]                : Iden to lift nodes by. May be specified multiple times.
```

Example:

- Lift the node with node ID 20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f:

```
storm> iden 20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f
inet:ipv4=1.2.3.4
        :type = unicast
        .created = 2024/04/19 14:26:24.535
```

intersect

The `intersect` command returns the intersection of the results from performing a pivot and/or traversal operation on multiple inbound nodes. In other words, `intersect` will return the subset of results that are **common** to each of the inbound nodes.

Syntax:

```
storm> intersect --help


    Yield an intersection of the results of running inbound nodes through a pivot.

    NOTE:
        This command must consume the entire inbound stream to produce the intersection.
        This type of stream consuming before yielding results can cause the query to␣
↪appear
        laggy in comparison with normal incremental stream operations.
```

(continues on next page)

(continued from previous page)

```
    Examples:

        // Show the it:mitre:attack:technique nodes common to several groups

        it:mitre:attack:group*in=(G0006, G0007) | intersect { ->
↪it:mitre:attack:technique }


Usage: intersect [options] <query>

Options:

  --help                      : Display the command usage.

Arguments:

  <query>                     : The pivot query to run each inbound node through.
```

layer

Storm includes `layer.*` commands that allow you to work with layers (see *Layer*).

- *layer.add*
- *layer.set*
- *layer.get*
- *layer.list*
- *layer.del*
- *layer.pull.add*
- *layer.pull.list*
- *layer.pull.del*
- *layer.push.add*
- *layer.push.list*
- *layer.push.del*

Help for individual `layer.*` commands can be displayed using:

```
<command> --help
```

layer.add

The `layer.add` command adds a layer to the Cortex.

Syntax

```
storm> layer.add --help

Add a layer to the cortex.

Usage: layer.add [options]

Options:

  --help                      : Display the command usage.
  --lockmemory                : Should the layer lock memory for performance.
  --readonly                  : Should the layer be readonly.
  --mirror <mirror>           : A telepath URL of an upstream layer/view to mirror.
  --growsize <growsize>       : Amount to grow the map size when necessary.
  --upstream <upstream>       : One or more telepath urls to receive updates from.
  --name <name>               : The name of the layer.
```

layer.set

The `layer.set` command sets an option for the specified layer.

Syntax

```
storm> layer.set --help

Set a layer option.

Usage: layer.set [options] <iden> <name> <valu>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Iden of the layer to modify.
  <name>                      : The name of the layer property to set.
  <valu>                      : The value to set the layer property to.
```

layer.get

The `layer.get` command retrieves the specified layer from a Cortex.

Syntax

```
storm> layer.get --help

Get a layer from the cortex.

Usage: layer.get [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  [iden]                      : Iden of the layer to get. If no iden is provided, the␣
↪main layer will be returned.
```

layer.list

The `layer.list` command lists the available layers in a Cortex.

Syntax

```
storm> layer.list --help

List the layers in the cortex.

Usage: layer.list [options]

Options:

  --help                      : Display the command usage.
```

layer.del

The `layer.del` command deletes a layer from a Cortex.

Syntax

```
storm> layer.del --help

Delete a layer from the cortex.

Usage: layer.del [options] <iden>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  <iden>                      : Iden of the layer to delete.
```

layer.pull.add

The `layer.pull.add` command adds a pull configuration to a layer.

Syntax

```
storm> layer.pull.add --help

Add a pull configuration to a layer.

Usage: layer.pull.add [options] <layr> <src>

Options:

  --help                      : Display the command usage.
  --offset <offset>           : Layer offset to begin pulling from (default: 0)

Arguments:

  <layr>                      : Iden of the layer to pull to.
  <src>                       : Telepath url of the source layer to pull from.
```

layer.pull.list

The `layer.pull.list` command lists the pull configurations for a layer.

Syntax

```
storm> layer.pull.list --help

Get a list of the pull configurations for a layer.

Usage: layer.pull.list [options] <layr>

Options:

  --help                      : Display the command usage.

Arguments:

  <layr>                      : Iden of the layer to retrieve pull configurations for.
```

layer.pull.del

The `layer.pull.del` command deletes a pull configuration from a layer.

Syntax

```
storm> layer.pull.del --help

Delete a pull configuration from a layer.

Usage: layer.pull.del [options] <layr> <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <layr>                      : Iden of the layer to modify.
  <iden>                      : Iden of the pull configuration to delete.
```

layer.push.add

The `layer.push.add` command adds a push configuration to a layer.

Syntax

```
storm> layer.push.add --help

Add a push configuration to a layer.

Usage: layer.push.add [options] <layr> <dest>

Options:

  --help                      : Display the command usage.
  --offset <offset>           : Layer offset to begin pushing from. (default: 0)

Arguments:

  <layr>                      : Iden of the layer to push from.
  <dest>                      : Telepath url of the layer to push to.
```

layer.push.list

The `layer.push.list` command lists the push configurations for a layer.

Syntax

```
storm> layer.push.list --help

Get a list of the push configurations for a layer.

Usage: layer.push.list [options] <layr>

Options:

  --help                      : Display the command usage.

Arguments:

  <layr>                      : Iden of the layer to retrieve push configurations for.
```

layer.push.del

The `layer.push.del` command deletes a push configuration from a layer.

Syntax

```
storm> layer.push.del --help

Delete a push configuration from a layer.

Usage: layer.push.del [options] <layr> <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <layr>                      : Iden of the layer to modify.
  <iden>                      : Iden of the push configuration to delete.
```

lift

Storm includes `lift.*` commands that allow you to perform specialized lift operations.

- *lift.byverb*

Help for individual `lift.*` commands can be displayed using:

```
<command> --help
```

lift.byverb

The `lift.byverb` command lifts nodes that are connected by the specified lightweight (light) edge. By default, the command lifts the N1 nodes (i.e., the nodes on the left side of the directional light edge relationship: `n1 -(<verb>)> n2`)

Note: For other commands associated with light edges, see `edges.del` and `model.edge.*` under *edges* and *model* respectively.

Syntax:

```
storm> lift.byverb --help


    Lift nodes from the current view by an light edge verb.

    Examples:

        # Lift all the n1 nodes for the light edge "foo"
        lift.byverb "foo"

        # Lift all the n2 nodes for the light edge "foo"
        lift.byverb --n2 "foo"

    Notes:

        Only a single instance of a node will be yielded from this command
        when that node is lifted via the light edge membership.


Usage: lift.byverb [options] <verb>

Options:

  --help                      : Display the command usage.
  --n2                        : Lift by the N2 value instead of N1 value.

Arguments:

  <verb>                      : The edge verb to lift nodes by.
```

limit

The `limit` command restricts the number of nodes returned from a given Storm query to the specified number of nodes.

Syntax:

```
storm> limit --help


    Limit the number of nodes generated by the query in the given position.
```

(continues on next page)

(continued from previous page)

```
    Example:

        inet:ipv4 | limit 10


Usage: limit [options] <count>

Options:

  --help                      : Display the command usage.

Arguments:

  <count>                     : The maximum number of nodes to yield.
```

Example:

- Lift a single IP address that FireEye associates with the threat group APT1:

```
storm> inet:ipv4#aka.feye.thr.apt1 | limit 1
```

Usage Notes:

- If the limit number specified (i.e., `limit 100`) is greater than the total number of nodes returned from the Storm query, no limit will be applied to the resultant nodes (i.e., all nodes will be returned).
- By design, `limit` imposes an artificial limit on the nodes returned by a query, which may impair effective analysis of data by restricting results. As such, `limit` is most useful for viewing a subset of a large result set or an exemplar node for a given form.
- While `limit` returns a sampling of nodes, it is not statistically random for the purposes of population sampling for algorithmic use.

macro

Note: See the *Storm Reference - Automation* guide for additional background on macros (as well as triggers and cron jobs), including examples.

Storm includes `macro.*` commands that allow you to work with macros (see *Macro*).

- *macro.list*
- *macro.set*
- *macro.get*
- *macro.exec*
- *macro.del*

Help for individual `macro.*` commands can be displayed using:

```
<command> --help
```

macro.list

The `macro.list` command lists the macros in a Cortex.

Syntax:

```
storm> macro.list --help


List the macros set on the cortex.


Usage: macro.list [options]

Options:

  --help                      : Display the command usage.
```

macro.set

The `macro.set` command creates (or modifies) a macro in a Cortex.

Syntax:

```
storm> macro.set --help


Set a macro definition in the cortex.

Variables can also be used that are defined outside the definition.

Examples:
    macro.set foobar ${ [+#foo] }

    # Use variable from parent scope
    macro.set bam ${ [ inet:ipv4=$val ] }
    $val=1.2.3.4 macro.exec bam


Usage: macro.set [options] <name> <storm>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the macro to set.
  <storm>                     : The storm command string or embedded query to set.
```

macro.get

The `macro.get` command retrieves and displays the specified macro.

Syntax:

```
storm> macro.get --help


Display the storm query for a macro in the cortex.


Usage: macro.get [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the macro to display.
```

macro.exec

The `macro.exec` command executes the specified macro.

Syntax:

```
storm> macro.exec --help


    Execute a named macro.

    Example:

        inet:ipv4#cno.threat.t80 | macro.exec enrich_foo



Usage: macro.exec [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the macro to execute
```

macro.del

The `macro.del` command deletes the specified macro from a Cortex.

Syntax:

```
storm> macro.del --help


Remove a macro definition from the cortex.


Usage: macro.del [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the macro to delete.
```

max

The `max` command returns the node from a given set that contains the highest value for a specified secondary property, tag interval, or variable.

Syntax:

```
storm> max --help


    Consume nodes and yield only the one node with the highest value for an expression.

    Examples:

        // Yield the file:bytes node with the highest :size property
        file:bytes#foo.bar | max :size

        // Yield the file:bytes node with the highest value for $tick
        file:bytes#foo.bar +.seen ($tick, $tock) = .seen | max $tick

        // Yield the it:dev:str node with the longest length
        it:dev:str | max $lib.len($node.value())



Usage: max [options] <valu>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  <valu>                      : The property or variable to use for comparison.
```

Examples:

- Return the DNS A record for woot.com with the most recent `.seen` value:

```
storm> inet:dns:a:fqdn=woot.com | max .seen
inet:dns:a=('woot.com', '107.21.53.159')
        :fqdn = woot.com
        :ipv4 = 107.21.53.159
        .created = 2024/04/19 14:26:25.064
        .seen = ('2014/08/13 00:00:00.000', '2014/08/14 00:00:00.000')
```

- Return the most recent WHOIS record for domain woot.com:

```
storm> inet:whois:rec:fqdn=woot.com | max :asof
inet:whois:rec=('woot.com', '2018/05/22 00:00:00.000')
        :asof = 2018/05/22 00:00:00.000
        :fqdn = woot.com
        :text = domain name: woot.com
        .created = 2024/04/19 14:26:25.138
```

merge

The `merge` command takes a subset of nodes from a forked view and merges them down to the next layer. The nodes can optionally be reviewed without actually merging them.

Contrast with *view.merge* for merging the entire contents of a forked view.

See the *view* and *layer* commands for working with views and layers.

Syntax:

```
storm> merge --help


    Merge edits from the incoming nodes down to the next layer.

    NOTE: This command requires the current view to be a fork.

    NOTE: The arguments for including/excluding tags can accept tag glob
          expressions for specifying tags. For more information on tag glob
          expressions, check the Synapse documentation for $node.globtags().

    Examples:

        // Having tagged a new #cno.mal.redtree subgraph in a forked view...

        #cno.mal.redtree | merge --apply

        // Print out what the merge command *would* do but dont.
```

(continues on next page)

(continued from previous page)

```
        #cno.mal.redtree | merge

        // Merge any org nodes with changes in the top layer.

        diff | +ou:org | merge --apply

        // Merge all tags other than cno.* from ou:org nodes with edits in the
        // top layer.

        diff | +ou:org | merge --only-tags --exclude-tags cno.** --apply

        // Merge only tags rep.vt.* and rep.whoxy.* from ou:org nodes with edits
        // in the top layer.

        diff | +ou:org | merge --include-tags rep.vt.* rep.whoxy.* --apply

        // Lift only inet:ipv4 nodes with a changed :asn property in top layer
        // and merge all changes.

        diff --prop inet:ipv4:asn | merge --apply

        // Lift only nodes with an added #cno.mal.redtree tag in the top layer and merge␣
↪them.

        diff --tag cno.mal.redtree | merge --apply


Usage: merge [options]

Options:

  --help                      : Display the command usage.
  --apply                     : Execute the merge changes.
  --no-tags                   : Do not merge tags/tagprops or syn:tag nodes.
  --only-tags                 : Only merge tags/tagprops or syn:tag nodes.
  --include-tags [<include_tags> ...]: Include specific tags/tagprops or syn:tag nodes␣
↪when merging, others are ignored. Tag glob expressions may be used to specify the tags.
↪ (default: [])
  --exclude-tags [<exclude_tags> ...]: Exclude specific tags/tagprops or syn:tag nodes␣
↪from merge.Tag glob expressions may be used to specify the tags. (default: [])
  --include-props [<include_props> ...]: Include specific props when merging, others are␣
↪ignored. (default: [])
  --exclude-props [<exclude_props> ...]: Exclude specific props from merge. (default: [])
  --diff                      : Enumerate all changes in the current layer.
```

min

The `min` command returns the node from a given set that contains the lowest value for a specified secondary property, tag interval, or variable.

Syntax:

```
storm> min --help


    Consume nodes and yield only the one node with the lowest value for an expression.

    Examples:

        // Yield the file:bytes node with the lowest :size property
        file:bytes#foo.bar | min :size

        // Yield the file:bytes node with the lowest value for $tick
        file:bytes#foo.bar +.seen ($tick, $tock) = .seen | min $tick

        // Yield the it:dev:str node with the shortest length
        it:dev:str | min $lib.len($node.value())



Usage: min [options] <valu>

Options:

  --help                      : Display the command usage.

Arguments:

  <valu>                      : The property or variable to use for comparison.
```

Examples:

- Return the DNS A record for woot.com with the oldest `.seen` value:

```
storm> inet:dns:a:fqdn=woot.com | min .seen
inet:dns:a=('woot.com', '75.101.146.4')
        :fqdn = woot.com
        :ipv4 = 75.101.146.4
        .created = 2024/04/19 14:26:25.073
        .seen = ('2013/09/21 00:00:00.000', '2013/09/22 00:00:00.000')
```

- Return the oldest WHOIS record for domain woot.com:

```
storm> inet:whois:rec:fqdn=woot.com | min :asof
inet:whois:rec=('woot.com', '2018/05/22 00:00:00.000')
        :asof = 2018/05/22 00:00:00.000
        :fqdn = woot.com
        :text = domain name: woot.com
        .created = 2024/04/19 14:26:25.138
```

model

Storm includes `model.*` commands that allow you to work with model elements.

`model.deprecated.*` commands allow you to view model elements (forms or properties) that have been marked as "deprecated", determine whether your Cortex contains deprecated nodes / nodes with deprecated properties, and optionally lock / unlock those properties to prevent (or allow) continued creation of deprecated model elements.

`model.edge.*` commands allow you to work with lightweight (light) edges. (See also the `edges.del` and `lift.byverb` commands under *edges* and *lift*, respectively.)

- *model.deprecated.check*
- *model.deprecated.lock*
- *model.deprecated.locks*
- *model.edge.list*
- *model.edge.set*
- *model.edge.get*
- *model.edge.del*

Help for individual `model.*` commands can be displayed using:

```
<command> --help
```

model.deprecated.check

The `model.deprecated.check` command lists deprecated elements, their lock status, and whether deprecated elements exist in the Cortex.

Syntax:

```
storm> model.deprecated.check --help

Check for lock status and the existence of deprecated model elements

Usage: model.deprecated.check [options]

Options:

  --help                      : Display the command usage.
```

model.deprecated.lock

The `model.deprecated.lock` command allows you to lock or unlock (e.g., disallow or allow the use of) deprecated model elements in a Cortex.

Syntax:

```
storm> model.deprecated.lock --help

Edit lock status of deprecated model elements.
```

(continues on next page)

(continued from previous page)

```
Usage: model.deprecated.lock [options] <name>

Options:

  --help                      : Display the command usage.
  --unlock                    : Unlock rather than lock the deprecated property.

Arguments:

  <name>                      : The deprecated form or property name to lock or * to␣
↪lock all.
```

model.deprecated.locks

The `model.deprecated.locks` command displays the lock status of all deprecated model elements.

Syntax:

```
storm> model.deprecated.locks --help

Display lock status of deprecated model elements.

Usage: model.deprecated.locks [options]

Options:

  --help                      : Display the command usage.
```

model.edge.list

The `model.edge.list` command displays the set of light edges currently defined in the Cortex and any `doc` values set on them.

Syntax:

```
storm> model.edge.list --help

List all edge verbs in the current view and their doc key (if set).

Usage: model.edge.list [options]

Options:

  --help                      : Display the command usage.
```

model.edge.set

The `model.edge.set` command allows you to set the value of a given key on a light edge (such as a `doc` value to specify a definition for the light edge). The current list of valid keys include the following:

- doc

Syntax:

```
storm> model.edge.set --help

Set a key-value for an edge verb that exists in the current view.

Usage: model.edge.set [options] <verb> <key> <valu>

Options:

  --help                      : Display the command usage.

Arguments:

  <verb>                      : The edge verb to add a key to.
  <key>                       : The key name (e.g. doc).
  <valu>                      : The string value to set.
```

model.edge.get

The `model.edge.get` command allows you to retrieve all of the keys that have been set on a light edge.

Syntax:

```
storm> model.edge.get --help

Retrieve key-value pairs for an edge verb in the current view.

Usage: model.edge.get [options] <verb>

Options:

  --help                      : Display the command usage.

Arguments:

  <verb>                      : The edge verb to retrieve.
```

model.edge.del

The `model.edge.del` command allows you to delete the key from a light edge (such as a `doc` property to specify a definition for the light edge). Deleting a key from a specific light edge does not delete the key from Synapse (e.g., the property can be re-added to the light edge or to other light edges).

Syntax:

```
storm> model.edge.del --help

Delete a global key-value pair for an edge verb in the current view.

Usage: model.edge.del [options] <verb> <key>

Options:

  --help                      : Display the command usage.

Arguments:

  <verb>                      : The edge verb to delete documentation for.
  <key>                       : The key name (e.g. doc).
```

movenodes

The `movenodes` command allows you to move nodes between layers (*Layer*) in a Cortex.

The command will move the specified storage nodes (see *Node, Storage*) - "sodes" for short - to the target layer. If a sode is the "left hand" (`n1`) of two nodes joined by a light edge (`n1 -(*)> n2`), then the edge is also moved.

Sodes are fully removed from the source layer(s) and added to (or merged with existing nodes in) the target layer. The history of the node (i.e., changes to the node, timestamps, associated user) in the **source** layer is preserved; the changes written to the **target** layer are owned by the user executing the `movenodes` command.

By default (i.e., if you do not specify a source and / or target layer), `movenodes` will migrate sodes from the bottom layer in the view, through each intervening layer (if any), and finally into the top layer. If you explicitly specify a source and target layer, `movenodes` migrates the sodes **directly** from the source to the target, skipping any intervening layers (if any).

Similarly, by default as the node is moved "up", any data for that node (property values, tags) in the higher layer will take precedence over (overwrite) data from a lower layer. This precedence behavior can be modified with the appropriate command switch.

The `movenodes` command is intended for use in the same layer stack. See the *copyto* command to copy nodes from a view to the write layer in a specified target view.

Note: The *merge* command specifically moves (merges) nodes from the top layer in a *View* to the underlying layer. Merging is a common **user action** performed in a standard "fork and merge" workflow. The `merge` command should be used to move/merge nodes **down** from a higher layer/view to a lower/underlying one.

The `movenodes` command allows you to move nodes between arbitrary layers and is meant to be used by Synapse **administrators** in very specific use cases (e.g., data that was accidentally merged into a lower layer that should not be there). It can be used to move nodes "up" from a lower layer to a higher one.

Syntax:

```
storm> movenodes --help


    Move storage nodes between layers.

    Storage nodes will be removed from the source layers and the resulting
    storage node in the destination layer will contain the merged values (merged
    in bottom up layer order by default).

    Examples:

        // Move storage nodes for ou:org nodes to the top layer

        ou:org | movenodes --apply

        // Print out what the movenodes command *would* do but dont.

        ou:org | movenodes

        // In a view with many layers, only move storage nodes from the bottom layer
        // to the top layer.

        $layers = $lib.view.get().layers
        $top = $layers.0.iden
        $bot = $layers."-1".iden

        ou:org | movenodes --srclayers $bot --destlayer $top

        // In a view with many layers, move storage nodes to the top layer and
        // prioritize values from the bottom layer over the other layers.

        $layers = $lib.view.get().layers
        $top = $layers.0.iden
        $mid = $layers.1.iden
        $bot = $layers.2.iden

        ou:org | movenodes --precedence $bot $top $mid


Usage: movenodes [options]

Options:

  --help                      : Display the command usage.
  --apply                     : Execute the move changes.
  --srclayers [<srclayers> ...]: Specify layers to move storage nodes from (defaults to
↪all below the top layer) (default: None)
  --destlayer <destlayer>     : Layer to move storage nodes to (defaults to the top
↪layer) (default: None)
  --precedence [<precedence> ...]: Layer precedence for resolving conflicts (defaults to
↪bottom up) (default: None)
```

movetag

The `movetag` command moves a Synapse tag and its associated tag tree from one location in a tag hierarchy to another location. It is equivalent to "renaming" a given tag and all of its subtags. Moving a tag consists of:

- Creating the new `syn:tag` node(s).
- Copying the definitions (`:title` and `:doc` properties) from the old `syn:tag` node to the new `syn:tag` node.
- Applying the new tag(s) to the nodes with the old tag(s).
  - If the old tag(s) have associated timestamps / time intervals, they will be applied to the new tag(s).
- Deleting the old tag(s) from the nodes.
- Setting the `:isnow` property of the old `syn:tag` node(s) to reference the new `syn:tag` node.
  - The old `syn:tag` nodes are **not** deleted.
  - Once the `:isnow` property is set, attempts to apply the old tag will automatically result in the new tag being applied.

See also the *tag* command.

Syntax:

```
storm> movetag --help


    Rename an entire tag tree and preserve time intervals.

    Example:

        movetag foo.bar baz.faz.bar


Usage: movetag [options] <oldtag> <newtag>

Options:

  --help                      : Display the command usage.

Arguments:

  <oldtag>                    : The tag tree to rename.
  <newtag>                    : The new tag tree name.
```

Examples:

- Move the tag named #research to #internal.research:

```
storm> movetag research internal.research
moved tags on 1 nodes.
```

- Move the tag tree #aka.fireeye.malware to #rep.feye.mal:

```
storm> movetag aka.fireeye.malware rep.feye.mal
moved tags on 1 nodes.
```

Usage Notes:

> **Warning:** `movetag` should be used with caution as when used incorrectly it can result in "deleted" (inadvertently moved / removed) or orphaned (inadvertently retained) tags. For example, in the second example query above, all `aka.fireeye.malware` tags are renamed `rep.feye.mal`, but the tag `aka.fireeye` still exists and is still applied to all of the original nodes. In other words, the result of the above command will be that nodes previously tagged `aka.fireeye.malware` will now be tagged both `rep.feye.mal` **and** `aka.fireeye`. Users may wish to test the command on sample data first to understand its effects before applying it in a production Cortex.

nodes

Storm includes `nodes.*` commands that allow you to work with nodes and `.nodes` files.

- *nodes.import*

Help for individual `nodes.*` commands can be displayed using:

```
<command> --help
```

nodes.import

The `nodes.import` command will import a Synapse `.nodes` file (i.e., a file containing a set / subgraph of nodes, light edges, and / or tags exported from a Cortex) from a specified URL.

Syntax:

```
storm> nodes.import --help

Import a nodes file hosted at a URL into the cortex. Yields created nodes.

Usage: nodes.import [options] <urls>

Options:

  --help                       : Display the command usage.
  --no-ssl-verify              : Ignore SSL certificate validation errors.

Arguments:

  [<urls> ...]                 : URL(s) to fetch nodes file from
```

note

Storm includes `note.*` commands that allow you to work with free form text notes (`meta:note` nodes).

- *note.add*

Help for individual `note.*` commands can be displayed using:

```
<command> --help
```

note.add

The `note.add` command will create a `meta:note` node containing the specified text and link it to the inbound node(s) via an `-(about)>` light edge (i.e., `meta:note=<guid> -(about)> <node(s)>`).

Syntax:

```
storm> note.add --help

Add a new meta:note node and link it to the inbound nodes using an -(about)> edge.

Usage: note.add [options] <text>

Options:

  --help                      : Display the command usage.
  --type <type>               : The note type.
  --yield                     : Yield the newly created meta:note node.

Arguments:

  <text>                      : The note text to add to the nodes.
```

Usage Notes:

Note: Synapse's data and analytical models are meant to represent a broad range of data and information in a structured (and therefore **queryable**) way. As free form notes are counter to this structured approach, we recommend using `meta:note` nodes as an exception rather than a regular practice.

once

The `once` command is used to ensure a given node is processed by the associated Storm command only once, even if the same command is executed in a different, independent Storm query. The `once` command uses *Node Data* to keep track of the associated Storm command's execution, so `once` is specific to the *View* in which it is executed. You can override the single-execution feature of `once` with the `--asof` parameter.

Syntax:

```
storm> once --help


    The once command is used to filter out nodes which have already been processed
    via the use of a named key. It includes an optional parameter to allow the node
    to pass the filter again after a given amount of time.

    For example, to run an enrichment command on a set of nodes just once:

        file:bytes#my.files | once enrich:foo | enrich.foo

    The once command filters out any nodes which have previously been through any other
    use of the "once" command using the same <name> (in this case "enrich:foo").
```

(continues on next page)

(continued from previous page)

```
    You may also specify the --asof option to allow nodes to pass the filter after a␣
↪given
    amount of time. For example, the following command will allow any given node through
    every 2 days:

        file:bytes#my.files | once enrich:foo --asof "-2 days" | enrich.foo

    Use of "--asof now" or any future date or positive relative time offset will always
    allow the node to pass the filter.

    State tracking data for the once command is stored as nodedata which is stored in␣
↪your
    view's write layer, making it view-specific. So if you have two views, A and B, and␣
↪they
    do not share any layers between them, and you execute this query in view A:

        inet:ipv4=8.8.8.8 | once enrich:address | enrich.baz

    And then you run it in view B, the node will still pass through the once command to␣
↪the
    enrich.baz portion of the query because the tracking data for the once command does␣
↪not
    yet exist in view B.


Usage: once [options] <name>

Options:

  --help                      : Display the command usage.
  --asof <asof>               : The associated time the name was updated/performed.␣
↪(default: None)

Arguments:

  <name>                      : Name of the action to only perform once.
```

parallel

The Storm `parallel` command allows you to execute a Storm query using a specified number of query pipelines. This can improve performance for some queries.

See also *background*.

Syntax:

```
storm> parallel --help


    Execute part of a query pipeline in parallel.
    This can be useful to minimize round-trip delay during enrichments.
```

(continues on next page)

(continued from previous page)

```
    Examples:
        inet:ipv4#foo | parallel { $place = $lib.import(foobar).lookup(:latlong) [␣
↪:place=$place ] }

    NOTE: Storm variables set within the parallel query pipelines do not interact.


Usage: parallel [options] <query>

Options:

  --help                      : Display the command usage.
  --size <size>               : The number of parallel Storm pipelines to execute.␣
↪(default: 8)

Arguments:

  <query>                     : The query to execute in parallel.
```

pkg

Storm includes `pkg.*` commands that allow you to work with Storm packages (see *Package*).

- *pkg.list*
- *pkg.load*
- *pkg.del*
- *pkg.docs*
- *pkg.perms.list*

Help for individual `pkg.*` commands can be displayed using:

```
<command> --help
```

Packages typically contain Storm commands and Storm library code used to implement a Storm *Service*.

pkg.list

The `pkg.list` command lists each Storm package loaded in the Cortex. Output is displayed in tabular form and includes the package name and version information.

Syntax:

```
storm> pkg.list --help

List the storm packages loaded in the cortex.

Usage: pkg.list [options]

Options:

  --help                      : Display the command usage.
```

pkg.load

The `pgk.load` command loads the specified package into the Cortex.

Syntax:

```
storm> pkg.load --help

Load a storm package from an HTTP URL.

Usage: pkg.load [options] <url>

Options:

  --help                      : Display the command usage.
  --raw                       : Response JSON is a raw package definition without an␣
↪envelope.
  --verify                    : Enforce code signature verification on the storm package.
  --ssl-noverify              : Specify to disable SSL verification of the server.

Arguments:

  <url>                       : The HTTP URL to load the package from.
```

pkg.del

The `pkg.del` command removes a Storm package from the Cortex.

Syntax:

```
storm> pkg.del --help

Remove a storm package from the cortex.

Usage: pkg.del [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name (or name prefix) of the package to remove.
```

pkg.docs

The `pkg.docs` command displays the documentation for a Storm package.

Syntax:

```
storm> pkg.docs --help

Display documentation included in a storm package.

Usage: pkg.docs [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name (or name prefix) of the package.
```

pkg.perms.list

The `pkg.perms.list` command lists the permissions declared by a Storm package.

Syntax:

```
storm> pkg.perms.list --help

List any permissions declared by the package.

Usage: pkg.perms.list [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name (or name prefix) of the package.
```

ps

Storm includes `ps.*` commands that allow you to work with Storm tasks/queries.

- *ps.list*
- *ps.kill*

Help for individual `ps.*` commands can be displayed using:

```
<command> --help
```

ps.list

The `ps.list` command lists the currently executing tasks/queries. By default, the command displays the first 120 characters of the executing query. The `--verbose` option can be used to display the full query regardless of length.

Syntax:

```
storm> ps.list --help

List running tasks in the cortex.

Usage: ps.list [options]

Options:

  --help                      : Display the command usage.
  --verbose                   : Enable verbose output.
```

ps.kill

The `ps.kill` command can be used to terminate an executing task/query. The command requires the *Iden* of the task to be terminated, which can be obtained with *ps.list*.

Syntax:

```
storm> ps.kill --help

Kill a running task/query within the cortex.

Usage: ps.kill [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid process iden␣
↪is accepted.
```

queue

Storm includes `queue.*` commands that allow you to work with queues (see *Queue*).

- *queue.add*
- *queue.list*
- *queue.del*

Help for individual `queue.*` commands can be displayed using:

```
<command> --help
```

queue.add

The `queue.add` command adds a queue to the Cortex.

Syntax:

```
storm> queue.add --help

Add a queue to the cortex.

Usage: queue.add [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the new queue.
```

queue.list

The `queue.list` command lists each queue in the Cortex.

Syntax:

```
storm> queue.list --help

List the queues in the cortex.

Usage: queue.list [options]

Options:

  --help                      : Display the command usage.
```

queue.del

The `queue.del` command removes a queue from the Cortex.

Syntax:

```
storm> queue.del --help

Remove a queue from the cortex.

Usage: queue.del [options] <name>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  <name>                      : The name of the queue to remove.
```

reindex

The `reindex` command is currently reserved for future use.

The intended purpose of this administrative command is to reindex a given node property. This may be necessary as part of a manual data migration.

Note: Any changes to the Synapse data model are noted in the changelog for the relevant Synapse release. Changes that require data migration are specifically noted and the data migration is typically performed automatically when deploying the new version. See the *Data Migration* section of the *Synapse Devops Guide* for additional detail.

Syntax:

```
storm> reindex --help


    Use admin privileges to re index/normalize node properties.

    NOTE: Currently does nothing but is reserved for future use.


Usage: reindex [options]

Options:

  --help                      : Display the command usage.
```

runas

The `runas` command allows you to execute a Storm query as a specified user.

Note: The `runas` commmand requires **admin** permisisons.

Syntax:

```
storm> runas --help


    Execute a storm query as a specified user.

    NOTE: This command requires admin privileges.

    Examples:

```

(continues on next page)

(continued from previous page)

```
        // Create a node as another user.
        runas someuser { [ inet:fqdn=foo.com ] }


Usage: runas [options] <user> <storm>

Options:

  --help                       : Display the command usage.
  --asroot                     : Propagate asroot to query subruntime.

Arguments:

  <user>                       : The user name or iden to execute the storm query as.
  <storm>                      : The storm query to execute.
```

scrape

The `scrape` command parses one or more secondary properties of the inbound node(s) and attempts to identify ("scrape") common forms from the content, creating the nodes if they do not already exist. This is useful (for example) for extracting forms such as email addresses, domains, URLs, hashes, etc. from unstructured text.

The `--refs` switch can be used to optionally link the source nodes(s) to the scraped forms via `refs` light edges.

By default, the `scrape` command will return the nodes that it received as input. The `--yield` option can be used to return the scraped nodes rather than the input nodes.

Syntax:

```
storm> scrape --help


    Use textual properties of existing nodes to find other easily recognizable nodes.

    Examples:

        # Scrape properties from inbound nodes and create standalone nodes.
        inet:search:query | scrape

        # Scrape properties from inbound nodes and make refs light edges to the scraped␣
↪nodes.
        inet:search:query | scrape --refs

        # Scrape only the :engine and :text props from the inbound nodes.
        inet:search:query | scrape :text :engine

        # Scrape properties inbound nodes and yield newly scraped nodes.
        inet:search:query | scrape --yield

        # Skip re-fanging text before scraping.
        inet:search:query | scrape --skiprefang
```

(continues on next page)

(continued from previous page)

```
        # Limit scrape to specific forms.
        inet:search:query | scrape --forms (inet:fqdn, inet:ipv4)


Usage: scrape [options] <values>

Options:

  --help                     : Display the command usage.
  --refs                     : Create refs light edges to any scraped nodes from the␣
↪input node
  --yield                    : Include newly scraped nodes in the output
  --skiprefang               : Do not remove de-fanging from text before scraping
  --forms <forms>            : Only scrape values which match specific forms. (default:␣
↪[])

Arguments:

  [<values> ...]             : Specific relative properties or variables to scrape
```

Example:

- Scrape the text of WHOIS records for the domain `woot.com` and create nodes for common forms found in the text:

```
storm> inet:whois:rec:fqdn=woot.com | scrape :text
inet:whois:rec=('woot.com', '2018/05/22 00:00:00.000')
        :asof = 2018/05/22 00:00:00.000
        :fqdn = woot.com
        :text = domain name: woot.com
        .created = 2024/04/19 14:26:25.138
```

Usage Notes:

- If no properties to scrape are specified, `scrape` will attempt to scrape **all** properties of the inbound nodes by default.
- `scrape` will only scrape node **properties**; it will not scrape files (this includes files that may be referenced by properties, such as `media:news:file`). In other words, `scrape` cannot be used to parse indicators from a file such as a PDF.
- `scrape` extracts the following forms / indicators (note that this list may change as the command is updated):
  - FQDNs
  - IPv4s
  - Servers (IPv4 / port combinations)
  - Hashes (MD5, SHA1, SHA256)
  - URLs
  - Email addresses
  - Cryptocurrency addresses

- `scrape` is able to recognize and account for common "defanging" techniques (such as `evildomain[.]com`, `myemail[@]somedomain.net`, or `hxxp://badwebsite.org/`), and will scrape "defanged" indicators by default. Use the `--skiprefang` switch to ignore defanged indicators.

service

Storm includes `service.*` commands that allow you to work with Storm services (see *Service*).

- *service.add*
- *service.list*
- *service.del*

Help for individual `service.*` commands can be displayed using:

    <command> --help

service.add

The `service.add` command adds a Storm service to the Cortex.

Syntax:

```
storm> service.add --help

Add a storm service to the cortex.

Usage: service.add [options] <name> <url>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The name of the service.
  <url>                       : The telepath URL for the remote service.
```

service.list

The `service.list` command lists each Storm service in the Cortex.

Syntax:

```
storm> service.list --help

List the storm services configured in the cortex.

Usage: service.list [options]

Options:

  --help                      : Display the command usage.
```

service.del

The `service.del` command removes a Storm service from the Cortex.

Syntax:

```
storm> service.del --help

Remove a storm service from the cortex.

Usage: service.del [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : The service identifier or prefix.
```

sleep

The `sleep` command adds a delay in returning each result for a given Storm query. By default, query results are streamed back and displayed as soon as they arrive for optimal performance. A `sleep` delay effectively slows the display of results.

Syntax:

```
storm> sleep --help


    Introduce a delay between returning each result for the storm query.

    NOTE: This is mostly used for testing / debugging.

    Example:

        #foo.bar | sleep 0.5



Usage: sleep [options] <delay>

Options:

  --help                      : Display the command usage.

Arguments:

  <delay>                     : Delay in floating point seconds.
```

Example:

- Retrieve email nodes from a Cortex every second:

```
storm> inet:email | sleep 1.0
inet:email=bar@gmail.com
        :fqdn = gmail.com
        :user = bar
        .created = 2024/04/19 14:26:26.047
inet:email=baz@gmail.com
        :fqdn = gmail.com
        :user = baz
        .created = 2024/04/19 14:26:26.052
inet:email=foo@gmail.com
        :fqdn = gmail.com
        :user = foo
        .created = 2024/04/19 14:26:26.040
```

spin

The `spin` command is used to suppress the output of a Storm query. `Spin` simply consumes all nodes sent to the command, so no nodes are output to the CLI. This allows you to execute a Storm query and view messages and results without displaying the associated nodes.

Syntax:

```
storm> spin --help


    Iterate through all query results, but do not yield any.
    This can be used to operate on many nodes without returning any.

    Example:

        foo:bar:size=20 [ +#hehe ] | spin



Usage: spin [options]

Options:

  --help                      : Display the command usage.
```

Example:

- Add the tag #int.research to any domain containing the string "firefox" but do not display the nodes.

```
storm> inet:fqdn~=firefox [+#int.research] | spin
```

stats

Storm includes `stats.*` commands that allow you to query and work with statistics.

- *stats.countby*

Help for individual `stats.*` commands can be displayed using:

```
<command> --help
```

stats.countby

The `stats.countby` command allows you to query and display a bar chart of tallied data in the Storm CLI.

Syntax:

```
storm> stats.countby --help


    Tally occurrences of values and display a bar chart of the results.

    Examples:

        // Show counts of geo:name values referenced by media:news nodes.
        media:news -(refs)> geo:name | stats.countby

        // Show counts of ASN values in a set of IPs.
        inet:ipv4#myips | stats.countby :asn

        // Show counts of attacker names for risk:compromise nodes.
        risk:compromise | stats.countby :attacker::name


Usage: stats.countby [options] <valu>

Options:

  --help                      : Display the command usage.
  --reverse                   : Display results in ascending instead of descending order.
  --size <size>               : Maximum number of bars to display. (default: None)
  --char <char>               : Character to use for bars. (default: #)
  --bar-width <bar_width>     : Width of the bars to display. (default: 50)
  --label-max-width <label_max_width>: Maximum width of the labels to display. (default:␣
↪None)
  --yield                     : Yield inbound nodes.
  --by-name                   : Print stats sorted by name instead of count.

Arguments:

  [valu]                      : A relative property or variable to tally.
```

tag

Storm includes `tag.*` commands that allow you to work with tags (see *Tag*).

- *tag.prune*

Help for individual `tag.*` commands can be displayed using:

```
<command> --help
```

See also the related *movetag* command.

tag.prune

The `tag.prune` command will delete the tags from incoming nodes, as well as all of their parent tags that don't have other tags as children.

Syntax:

```
storm> tag.prune --help


    Prune a tag (or tags) from nodes.

    This command will delete the tags specified as parameters from incoming nodes,
    as well as all of their parent tags that don't have other tags as children.

    For example, given a node with the tags:

        #parent
        #parent.child
        #parent.child.grandchild

    Pruning the parent.child.grandchild tag would remove all tags. If the node had
    the tags:

        #parent
        #parent.child
        #parent.child.step
        #parent.child.grandchild

    Pruning the parent.child.grandchild tag will only remove the parent.child.grandchild
    tag as the parent tags still have other children.

    Examples:

        # Prune the parent.child.grandchild tag
        inet:ipv4=1.2.3.4 | tag.prune parent.child.grandchild


Usage: tag.prune [options] <tags>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  [<tags> ...]                : Names of tags to prune.
```

tee

The `tee` command executes multiple Storm queries on the inbound nodes and returns the combined result set.

Syntax:

```
storm> tee --help


    Execute multiple Storm queries on each node in the input stream, joining output␣
↪streams together.

    Commands are executed in order they are given; unless the ``--parallel`` switch is␣
↪provided.

    Examples:

        # Perform a pivot out and pivot in on a inet:ivp4 node
        inet:ipv4=1.2.3.4 | tee { -> * } { <- * }

        # Also emit the inbound node
        inet:ipv4=1.2.3.4 | tee --join { -> * } { <- * }

        # Execute multiple enrichment queries in parallel.
        inet:ipv4=1.2.3.4 | tee -p { enrich.foo } { enrich.bar } { enrich.baz }



Usage: tee [options] <query>

Options:

  --help                      : Display the command usage.
  --join                      : Emit inbound nodes after processing storm queries.
  --parallel                  : Run the storm queries in parallel instead of sequence.␣
↪The node output order is not guaranteed.

Arguments:

  [<query> ...]               : Specify a query to execute on the input nodes.
```

Examples:

- Return the set of domains and IP addresses associated with a set of DNS A records.

```
storm> inet:fqdn:zone=mydomain.com -> inet:dns:a | tee { -> inet:fqdn } { -> inet:ipv4 }
inet:fqdn=baz.mydomain.com
```

(continues on next page)

(continued from previous page)

```
        :domain = mydomain.com
        :host = baz
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.263
inet:ipv4=127.0.0.2
        :type = loopback
        .created = 2024/04/19 14:26:29.263
inet:fqdn=foo.mydomain.com
        :domain = mydomain.com
        :host = foo
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.248
inet:ipv4=8.8.8.8
        :type = unicast
        .created = 2024/04/19 14:26:29.248
inet:fqdn=bar.mydomain.com
        :domain = mydomain.com
        :host = bar
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.256
inet:ipv4=34.56.78.90
        :type = unicast
        .created = 2024/04/19 14:26:29.256
```

- Return the set of domains and IP addresses associated with a set of DNS A records along with the original DNS A records.

```
storm> inet:fqdn:zone=mydomain.com -> inet:dns:a | tee --join { -> inet:fqdn } { ->␣
↪inet:ipv4 }
inet:fqdn=baz.mydomain.com
        :domain = mydomain.com
        :host = baz
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.263
inet:ipv4=127.0.0.2
        :type = loopback
        .created = 2024/04/19 14:26:29.263
inet:dns:a=('baz.mydomain.com', '127.0.0.2')
        :fqdn = baz.mydomain.com
        :ipv4 = 127.0.0.2
        .created = 2024/04/19 14:26:29.263
inet:fqdn=foo.mydomain.com
        :domain = mydomain.com
        :host = foo
```

(continues on next page)

(continued from previous page)

```
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.248
inet:ipv4=8.8.8.8
        :type = unicast
        .created = 2024/04/19 14:26:29.248
inet:dns:a=('foo.mydomain.com', '8.8.8.8')
        :fqdn = foo.mydomain.com
        :ipv4 = 8.8.8.8
        .created = 2024/04/19 14:26:29.248
inet:fqdn=bar.mydomain.com
        :domain = mydomain.com
        :host = bar
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:26:29.256
inet:ipv4=34.56.78.90
        :type = unicast
        .created = 2024/04/19 14:26:29.256
inet:dns:a=('bar.mydomain.com', '34.56.78.90')
        :fqdn = bar.mydomain.com
        :ipv4 = 34.56.78.90
        .created = 2024/04/19 14:26:29.256
```

Usage Notes:

- `tee` can take an arbitrary number of Storm queries (i.e., 1 to n queries) as arguments.

tree

The `tree` command recursively performs the specified pivot until no additional nodes are returned.

Syntax:

```
storm> tree --help


    Walk elements of a tree using a recursive pivot.

    Examples:

        # pivot upward yielding each FQDN
        inet:fqdn=www.vertex.link | tree { :domain -> inet:fqdn }


Usage: tree [options] <query>

Options:

  --help                      : Display the command usage.
```

(continues on next page)

(continued from previous page)

```
Arguments:

  <query>                     : The pivot query
```

Example:

- List the full set of tags in the "TTP" tag hierarchy.

```
storm> syn:tag=ttp | tree { $node.value() -> syn:tag:up }
syn:tag=ttp
        :base = ttp
        :depth = 0
        .created = 2024/04/19 14:26:29.396
syn:tag=ttp.phish
        :base = phish
        :depth = 1
        :up = ttp
        .created = 2024/04/19 14:26:29.407
syn:tag=ttp.phish.payload
        :base = payload
        :depth = 2
        :up = ttp.phish
        .created = 2024/04/19 14:26:29.407
syn:tag=ttp.opsec
        :base = opsec
        :depth = 1
        :up = ttp
        .created = 2024/04/19 14:26:29.396
syn:tag=ttp.opsec.anon
        :base = anon
        :depth = 2
        :up = ttp.opsec
        .created = 2024/04/19 14:26:29.396
syn:tag=ttp.se
        :base = se
        :depth = 1
        :up = ttp
        .created = 2024/04/19 14:26:29.402
syn:tag=ttp.se.masq
        :base = masq
        :depth = 2
        :up = ttp.se
        .created = 2024/04/19 14:26:29.402
```

Usage Notes:

- `tree` is useful for "walking" a set of properties with a single command vs. performing an arbitrary number of pivots until the end of the data is reached.

trigger

Note: See the *Storm Reference - Automation* guide for additional background on triggers (as well as cron jobs and macros), including examples.

Storm includes `trigger.*` commands that allow you to create automated event-driven triggers (see *Trigger*) using the Storm query syntax.

- *trigger.add*
- *trigger.list*
- *trigger.mod*
- *trigger.disable*
- *trigger.enable*
- *trigger.del*

Help for individual `trigger.*` commands can be displayed using:

```
<command> --help
```

Triggers are added to the Cortex as **runtime nodes** ("runt nodes" - see *Node, Runt*) of the form `syn:trigger`. These runt nodes can be lifted and filtered just like standard nodes in Synapse.

trigger.add

The `trigger.add` command adds a trigger to a Cortex.

Syntax:

```
storm> trigger.add --help


Add a trigger to the cortex.

Notes:
    Valid values for condition are:
        * tag:add
        * tag:del
        * node:add
        * node:del
        * prop:set
        * edge:add
        * edge:del

When condition is tag:add or tag:del, you may optionally provide a form name
to restrict the trigger to fire only on tags added or deleted from nodes of
those forms.

The added tag is provided to the query as an embedded variable '$tag'.

Simple one level tag globbing is supported, only at the end after a period,
that is aka.* matches aka.foo and aka.bar but not aka.foo.bar. aka* is not
```

(continues on next page)

(continued from previous page)

```
supported.

When the condition is edge:add or edge:del, you may optionally provide a
form name or a destination form name to only fire on edges added or deleted
from nodes of those forms.

Examples:
    # Adds a tag to every inet:ipv4 added
    trigger.add node:add --form inet:ipv4 --query {[ +#mytag ]}

    # Adds a tag #todo to every node as it is tagged #aka
    trigger.add tag:add --tag aka --query {[ +#todo ]}

    # Adds a tag #todo to every inet:ipv4 as it is tagged #aka
    trigger.add tag:add --form inet:ipv4 --tag aka --query {[ +#todo ]}

    # Adds a tag #todo to the N1 node of every refs edge add
    trigger.add edge:add --verb refs --query {[ +#todo ]}

    # Adds a tag #todo to the N1 node of every seen edge delete, provided that
    # both nodes are of form file:bytes
    trigger.add edge:del --verb seen --form file:bytes --n2form file:bytes --query {[ +
↪#todo ]}


Usage: trigger.add [options] <condition>

Options:

  --help                        : Display the command usage.
  --form <form>                 : Form to fire on.
  --tag <tag>                   : Tag to fire on.
  --prop <prop>                 : Property to fire on.
  --verb <verb>                 : Edge verb to fire on.
  --n2form <n2form>             : The form of the n2 node to fire on.
  --query <storm>               : Query for the trigger to execute.
  --async                       : Make the trigger run in the background.
  --disabled                    : Create the trigger in disabled state.
  --name <name>                 : Human friendly name of the trigger.
  --view <view>                 : The view to add the trigger to.

Arguments:

  <condition>                   : Condition for the trigger.
```

trigger.list

The `trigger-list` command displays the set of triggers in the Cortex that the current user can view / modify based on their permissions. Triggers are displayed at the Storm CLI in tabular format, with columns including the user who created the trigger, the *Iden* of the trigger, the condition that fires the trigger (i.e., `node:add`), and the Storm query associated with the trigger.

Triggers are displayed in alphanumeric order by iden. Triggers are sorted upon Cortex initialization, so newly-created triggers will be displayed at the bottom of the list until the list is re-sorted the next time the Cortex is restarted.

Note: Triggers can also be viewed in runt node form as `syn:trigger` nodes.

Syntax:

```
storm> trigger.list --help

List existing triggers in the cortex.

Usage: trigger.list [options]

Options:

  --help                      : Display the command usage.
  --all                       : List every trigger in every readable view, rather than␣
↪just the current view.
```

trigger.mod

The `trigger.mod` command modifies the Storm query associated with a specific trigger. To modify a trigger, you must provide the first portion of the trigger's iden (i.e., enough of the iden that the trigger can be uniquely identified), which can be obtained using `trigger.list` or by lifting the appropriate `syn:trigger` node.

Note: Other aspects of the trigger, such as the condition used to fire the trigger or the tag or property associated with the trigger, cannot be modified once the trigger has been created. To change these aspects, you must delete and re-add the trigger.

Syntax:

```
storm> trigger.mod --help

Modify an existing trigger's query.

Usage: trigger.mod [options] <iden> <query>

Options:

  --help                      : Display the command usage.

Arguments:

```

(continues on next page)

(continued from previous page)

```
  <iden>                      : Any prefix that matches exactly one valid trigger iden␣
↪is accepted.
  <query>                     : New storm query for the trigger.
```

trigger.disable

The `trigger.disable` command disables a trigger and prevents it from firing without removing it from the Cortex. To disable a trigger, you must provide the first portion of the trigger's iden (i.e., enough of the iden that the trigger can be uniquely identified), which can be obtained using `trigger.list` or by lifting the appropriate `syn:trigger` node.

Syntax:

```
storm> trigger.disable --help

Disable a trigger in the cortex.

Usage: trigger.disable [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid trigger iden␣
↪is accepted.
```

trigger.enable

The `trigger-enable` command enables a disabled trigger. To enable a trigger, you must provide the first portion of the trigger's iden (i.e., enough of the iden that the trigger can be uniquely identified), which can be obtained using `trigger.list` or by lifting the appropriate `syn:trigger` node.

Note: Triggers are enabled by default upon creation.

Syntax:

```
storm> trigger.enable --help

Enable a trigger in the cortex.

Usage: trigger.enable [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:
```

(continues on next page)

(continued from previous page)

```
  <iden>                      : Any prefix that matches exactly one valid trigger iden␣
↪is accepted.
```

trigger.del

The `trigger.del` command permanently removes a trigger from the Cortex. To delete a trigger, you must provide the first portion of the trigger's iden (i.e., enough of the iden that the trigger can be uniquely identified), which can be obtained using `trigger.list` or by lifting the appropriate `syn:trigger` node.

Syntax:

```
storm> trigger.del --help

Delete a trigger from the cortex.

Usage: trigger.del [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Any prefix that matches exactly one valid trigger iden␣
↪is accepted.
```

uniq

The `uniq` command removes duplicate results from a Storm query. Results are uniqued based on each node's node identifier (node ID / iden) so that only the first node with a given node ID is returned.

Syntax:

```
storm> uniq --help


    Filter nodes by their uniq iden values.
    When this is used a Storm pipeline, only the first instance of a
    given node is allowed through the pipeline.

    A relative property or variable may also be specified, which will cause
    this command to only allow through the first node with a given value for
    that property or value rather than checking the node iden.

    Examples:

        # Filter duplicate nodes after pivoting from inet:ipv4 nodes tagged with
↪#badstuff
        #badstuff +inet:ipv4 ->* | uniq

        # Unique inet:ipv4 nodes by their :asn property
```

(continues on next page)

(continued from previous page)

```
        #badstuff +inet:ipv4 | uniq :asn


Usage: uniq [options] <value>

Options:

  --help                      : Display the command usage.

Arguments:

  [value]                     : A relative property or variable to uniq by.
```

Examples:

- Lift all of the unique IP addresses that domains associated with the Fancy Bear threat group have resolved to:

```
storm> inet:fqdn#rep.threatconnect.fancybear -> inet:dns:a -> inet:ipv4 | uniq
inet:ipv4=111.90.148.124
        :type = unicast
        .created = 2024/04/19 14:26:29.668
inet:ipv4=209.99.40.222
        :type = unicast
        .created = 2024/04/19 14:26:29.677
inet:ipv4=141.8.224.221
        :type = unicast
        .created = 2024/04/19 14:26:29.685
```

uptime

The `uptime` command displays the uptime for the Cortex or specified service.

Syntax:

```
storm> uptime --help

Print the uptime for the Cortex or a connected service.

Usage: uptime [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  [name]                      : The name, or iden, of the service (if not provided␣
↪defaults to the Cortex).
```

vault

Storm includes `vault.*` commands that allow you to create and manage vaults (see *Vault*).

- *vault.add*
- *vault.list*
- *vault.set.configs*
- *vault.set.perm*
- *vault.set.secrets*
- *vault.del*

Help for individual `vault.*` commands can be displayed using:

```
<command> --help
```

vault.add

The `vault.add` command creates a new vault.

Syntax:

```
storm> vault.add --help


            Add a vault.

            Examples:

                // Add a global vault with type `synapse-test`
                vault.add "shared-global-vault" synapse-test ({'apikey': 'foobar'}) ({})␣
↪--global

                // Add a user vault with type `synapse-test`
                vault.add "visi-user-vault" synapse-test ({'apikey': 'barbaz'}) ({}) --
↪user visi

                // Add a role vault with type `synapse-test`
                vault.add "contributor-role-vault" synapse-test ({'apikey': 'bazquux'}) (
↪{}) --role contributor

                // Add an unscoped vault with type `synapse-test`
                vault.add "unscoped-vault" synapse-test ({'apikey': 'quuxquo'}) ({'server
↪': 'api.foobar.com'}) --unscoped visi


Usage: vault.add [options] <name> <type> <secrets> <configs>

Options:

  --help                      : Display the command usage.
  --user <user>               : This vault is a user-scoped vault, for the specified␣
↪user name.
```

(continues on next page)

(continued from previous page)

```
  --role <role>              : This vault is a role-scoped vault, for the specified␣
↪role name.
  --unscoped <unscoped>      : This vault is an unscoped vault, for the specified user␣
↪name.
  --global                   : This vault is a global-scoped vault.

Arguments:

  <name>                     : The vault name.
  <type>                     : The vault type.
  <secrets>                  : The secrets to store in the new vault.
  <configs>                  : The configs to store in the new vault.
```

vault.list

The `vault.list` command displays the available vaults.

Syntax:

```
storm> vault.list --help


            List available vaults.


Usage: vault.list [options]

Options:

  --help                     : Display the command usage.
  --name <name>              : Only list vaults with the specified name or iden.
  --type <type>              : Only list vaults with the specified type.
  --showsecrets              : Print vault secrets.
```

vault.set.configs

The `vault.set.configs` sets configuration options for the specified vault.

Syntax:

```
storm> vault.set.configs --help


            Set vault config data.

            Examples:

                // Set data to visi's user vault configs
                vault.set.configs "visi-user-vault" color --value orange
```

(continues on next page)

(continued from previous page)

```
                // Set data to contributor's role vault configs
                vault.set.configs "contributor-role-vault" color --value blue

                // Remove apikey from a global vault configs
                vault.set.configs "some-global-vault" color --delete


Usage: vault.set.configs [options] <name> <key>

Options:

  --help                      : Display the command usage.
  --value <value>             : The config value to store in the vault.
  --delete                    : Specify this flag to remove the config from the vault.

Arguments:

  <name>                      : The vault name or iden.
  <key>                       : The key for the config value.
```

vault.set.perm

The `vault.set.perm` command grants or revokes permissions to a vault.

Syntax:

```
storm> vault.set.perm --help


            Set permissions on a vault.

            Examples:

                // Give blackout read permissions to visi's user vault
                vault.set.perm "my-user-vault" blackout --level read

                // Give the contributor role read permissions to visi's user vault
                vault.set.perm "my-user-vault" --role contributor --level read

                // Revoke blackout's permissions from visi's user vault
                vault.set.perm "my-user-vault" blackout --revoke

                // Give visi read permissions to the contributor role vault. (Assume
                // visi is not a member of the contributor role).
                vault.set.perm "contributor-role-vault" visi read


Usage: vault.set.perm [options] <name>

Options:

```

(continues on next page)

(continued from previous page)

```
  --help                         : Display the command usage.
  --user <user>                  : The user name or role name to update in the vault.
  --role <role>                  : Specified when `user` is a role name.
  --level <level>                : The permission level to grant.
  --revoke                       : Specify this flag when revoking an existing permission.

Arguments:

  <name>                         : The vault name or iden to set permissions on.
```

vault.set.secrets

The `vault.set.secrets` command sets the specified secret for the vault.

Syntax:

```
storm> vault.set.secrets --help


            Set vault secret data.

            Examples:

                // Set data to visi's user vault secrets
                vault.set.secrets "visi-user-vault" apikey --value foobar

                // Set data to contributor's role vault secrets
                vault.set.secrets "contributor-role-vault" apikey --value barbaz

                // Remove apikey from a global vault secrets
                vault.set.secrets "some-global-vault" apikey --delete


Usage: vault.set.secrets [options] <name> <key>

Options:

  --help                         : Display the command usage.
  --value <value>                : The secret value to store in the vault.
  --delete                       : Specify this flag to remove the secret from the vault.

Arguments:

  <name>                         : The vault name or iden.
  <key>                          : The key for the secret value.
```

vault.del

The `vault.del` command deletes a vault.

Syntax:

```
storm> vault.del --help


            Delete a vault.

            Examples:

                // Delete visi's user vault
                vault.del "visi-user-vault"

                // Delete contributor's role vault
                vault.del "contributor-role-vault"


Usage: vault.del [options] <name>

Options:

  --help                      : Display the command usage.

Arguments:

  <name>                      : The vault name or iden.
```

version

The `version` command displays the current version of Synapse and associated metadata.

Syntax:

```
storm> version --help

Show version metadata relating to Synapse.

Usage: version [options]

Options:

  --help                      : Display the command usage.
```

view

Storm includes `view.*` commands that allow you to work with views (see *View*).

- *view.add*
- *view.fork*
- *view.set*
- *view.get*
- *view.list*
- *view.exec*
- *view.merge*
- *view.del*

Help for individual `view.*` commands can be displayed using:

```
<command> --help
```

view.add

The `view.add` command adds a view to the Cortex.

Syntax:

```
storm> view.add --help

Add a view to the cortex.

Usage: view.add [options]

Options:

  --help                      : Display the command usage.
  --name <name>               : The name of the new view. (default: None)
  --worldreadable <worldreadable>: Grant read access to the `all` role. (default: False)
  --layers [<layers> ...]     : Layers for the view. (default: [])
```

view.fork

The `view.fork` command forks an existing view from the Cortex. Forking a view creates a new view with a new writeable layer on top of the set of layers from the previous (forked) view.

Syntax:

```
storm> view.fork --help

Fork a view in the cortex.

Usage: view.fork [options] <iden>

Options:
```

(continues on next page)

(continued from previous page)

```
  --help                          : Display the command usage.
  --name <name>                   : Name for the newly forked view. (default: None)

Arguments:

  <iden>                          : Iden of the view to fork.
```

view.set

The `view.set` command sets a property on the specified view.

Syntax:

```
storm> view.set --help

Set a view option.

Usage: view.set [options] <iden> <name> <valu>

Options:

  --help                          : Display the command usage.

Arguments:

  <iden>                          : Iden of the view to modify.
  <name>                          : The name of the view property to set.
  <valu>                          : The value to set the view property to.
```

view.get

The `view.get` command retrieves an existing view from the Cortex.

Syntax:

```
storm> view.get --help

Get a view from the cortex.

Usage: view.get [options] <iden>

Options:

  --help                          : Display the command usage.

Arguments:

  [iden]                          : Iden of the view to get. If no iden is provided, the␣
↪main view will be returned.
```

view.list

The `view.list` command lists the views in the Cortex.

Syntax:

```
storm> view.list --help

List the views in the cortex.

Usage: view.list [options]

Options:

  --help                      : Display the command usage.
```

view.exec

The `view.exec` command executes a Storm query in the specified view.

Behavior and Limitations

The `view.exec` command creates its own execution environment (sub-runtime) to execute a Storm query in a different view. This results in a firm separation boundary between the source view and the destination view where nodes do not pass in or out across the `view.exec` boundary. Pipelines, events, messages, etc will NOT pass from the destination view to the source view or vice-versa. This includes `$lib.print(...)`, `$lib.warn(...)`, and other functions that may print to the CLI.

Variables declared before the `view.exec` are accessible in the destination view (including assignment). The interactive help example demonstrates this behavior:

```
// Move some tagged nodes to another view
inet:fqdn#foo.bar $fqdn=$node.value() | view.exec 95d5f31f0fb414d2b00069d3b1ee64c6 { [␣
↪inet:fqdn=$fqdn ] }
```

Here we have `inet:fqdn` nodes with the tag `#foo.bar` being lifted and their value (not the node) is saved into the `$fqdn` variable. This variable is later accessible in the `view.exec` sub-query and used to create an `inet:fqdn` node in the destination view. If more than one `inet:fqdn` node with the tag `#foo.bar` exists, the `view.exec` command would be executed once for each node in the pipeline as expected. Again, the actual nodes will not be accessible in the `view.exec` query. Also note the sub-query executed in the `view.exec` may assign a different value back to `$fqdn` to be accessed by the source view (that doesn't happen in this example though).

Inline functions are bound to the scope they are declared in. For `view.exec`, this means that a function declared outside the `view.exec` command will still run in the original scope/view, not the view specified to `view.exec`.

Syntax:

```
storm> view.exec --help


    Execute a storm query in a different view.

    NOTE: Variables are passed through but nodes are not. The behavior of this command␣
↪may be
    non-intuitive in relation to the way storm normally operates. For further␣
```

(continues on next page)

(continued from previous page)

```
↪information on
    behavior and limitations when using `view.exec`, reference the `view.exec` section␣
↪of the
    Synapse User Guide: https://v.vtx.lk/view-exec.

    Examples:

        // Move some tagged nodes to another view
        inet:fqdn#foo.bar $fqdn=$node.value() | view.exec␣
↪95d5f31f0fb414d2b00069d3b1ee64c6 { [ inet:fqdn=$fqdn ] }


Usage: view.exec [options] <view> <storm>

Options:

  --help                      : Display the command usage.

Arguments:

  <view>                      : The GUID of the view in which the query will execute.
  <storm>                     : The storm query to execute on the view.
```

view.merge

The `view.merge` command merges **all** data from a forked view into its parent view.

Contrast with *merge* which can merge a subset of nodes.

Syntax:

```
storm> view.merge --help

Merge a forked view into its parent view.

Usage: view.merge [options] <iden>

Options:

  --help                      : Display the command usage.
  --delete                    : Once the merge is complete, delete the layer and view.

Arguments:

  <iden>                      : Iden of the view to merge.
```

view.del

The `view.del` command permanently deletes a view from the Cortex.

Syntax:

```
storm> view.del --help

Delete a view from the cortex.

Usage: view.del [options] <iden>

Options:

  --help                      : Display the command usage.

Arguments:

  <iden>                      : Iden of the view to delete.
```

wget

The `wget` command retrieves content from one or more specified URLs. The command creates and yields `inet:urlfile` nodes and the retrieved content (`file:bytes`) is stored in the *Axon*.

Syntax:

```
storm> wget --help

Retrieve bytes from a URL and store them in the axon. Yields inet:urlfile nodes.

Examples:

    # Specify custom headers and parameters
    inet:url=https://vertex.link/foo.bar.txt | wget --headers ({"User-Agent": "Foo/Bar"}
↪) --params ({"clientid": "42"})

    # Download multiple URL targets without inbound nodes
    wget https://vertex.link https://vtx.lk


Usage: wget [options] <urls>

Options:

  --help                      : Display the command usage.
  --no-ssl-verify             : Ignore SSL certificate validation errors.
  --timeout <timeout>         : Configure the timeout for the download operation.␣
↪(default: 300)
  --params <params>           : Provide a dict containing url parameters. (default: None)
  --headers <headers>         : Provide a Storm dict containing custom request headers.␣
↪(default:
{                                 'Accept': '*/*',
```

(continues on next page)

(continued from previous page)

```
                                   'Accept-Encoding': 'gzip, deflate',
                                   'Accept-Language': 'en-US,en;q=0.9',
                                   'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) '
                                                 'AppleWebKit/537.36 (KHTML, like Gecko) '
                                                 'Chrome/92.0.4515.131 Safari/537.36'})
  --no-headers                : Do NOT use any default headers.

Arguments:

  [<urls> ...]                : URLs to download.
```

3.6.11 Storm Reference - Automation

Background

Synapse supports large-scale analysis over a broad range of data with speed and efficiency. Many features that support this analysis are built into Synapse's architecture, from performance-optimized indexing and storage to an extensible data model that allows you to reason over data in a structured manner.

Synapse also supports large-scale analysis through the use of **automation.** Synapse's automation features include:

- *Triggers and Cron*
- *Macros*
- *Dmons*

By making use of automation in Synapse, you can free analysts from performing tedious work and allow them to focus on more detailed analysis and complex tasks. You can also scale analytical operations by limiting the amount of work that must be performed manually.

Automation in Synapse uses the Storm query language: **anything that can be written in Storm can be automated,** from the simple to the more advanced. Actions performed through automation are limited only by imagination and Storm proficiency. Automation can be basic: "if X occurs, do Y" or "once a week, update Z". However, automation can also take advantage of all available Storm features (including subqueries, variables, libraries, and control flow logic), to support highly customized tasks and workflows.

Considerations

The following items should be taken into account when planning the use of automation in your environment.

Permissions

Permissions impact the use of automation in Synapse in various ways. In some cases, you must explicitly grant permission for users to create and manage automation. In other cases, the permissions that a given automated task runs under may vary based on the type of automation used. See the relevant sections below for additional detail.

Tip: For a detailed discussion of permissions in Synapse, refer to the Synapse Admin Guide.

Scope

Automation components vary with respect to where they reside and execute within Synapse; some elements are global (within a Cortex) while some reside and execute within a specific view. Organizations that leverage multiple views for their Synapse architecture or that make use of Synapse's fork and merge capabilities should refer to the sections below for information on how views and layers may impact automation.

Tip: For a more general discussion of views and layers, including a discussion of forking and merging views, see the *Views and Layers* section of the Synapse User Guide.

Testing

Always test your automation before putting it into production. Storm used in automation can be syntactically correct (that is, the query uses proper Storm), but contain logical errors (the query does not actually do what you want it to do). In addition, new automation may interact with existing automation in unexpected ways. Test your automation in a development environment (either a separate development instance, or a separate fork of your production view) before implementing it in production.

Use Cases

Organizations can implement automation as they see fit. Some automation may be enterprise-wide, used to support an organization's overall mission or analysis efforts. Other automation may be put in place by individual analysts to support their own research efforts, either on an ongoing or temporary basis.

Design

There are varying approaches for "how" to write and implement automation. For example:

- **Location of automation code.** The Storm code run by individual triggers and cron jobs can be written and stored as part of the automation itself. This approach helps keep automation "self-contained". However, it may provide less flexibility in executing the associated Storm compared with the use of macros.

  Alternatively, tasks such as triggers and cron jobs can be written to execute minimal Storm queries whose purpose is to call more extensive Storm stored centrally in macros. This approach consolidates much of the associated Storm, which may make it easier to manage and maintain. Storm placed in macros also provides flexibility as the macro can be called by a trigger, a cron job, or a user as part of a Storm query.

- **Size of automation.** Automation can be written as many small, individual elements. Each element can perform a relatively simple task, but the elements can work together like building blocks to orchestrate larger-scale operations. This approach keeps tasks "bite sized" and the Storm executed by a given piece of automation generally simpler. However it may result in a larger number of automation elements to maintain, and may make it more challenging to understand the potential interactions of so many different elements.

  Alternatively, automation can be implemented using fewer elements that perform larger, more unified tasks (or that consolidate numerous smaller tasks into a larger set of Storm code). This approach results in fewer automation elements overall, but typically requires you to write and maintain more advanced Storm (e.g., to create a small number of macros with switch or if/else statements to each manage a variety of tasks). However, the Storm is consolidated in a few locations, which may make managing and troubleshooting easier.

Each approach has its pros and cons; there is no single "right" way. In addition, you do not have to take an "either / or" approach; what works best in your environment or for a particular task will depend on your needs (and possibly some trial and error).

Governance / Management

Consider any oversight or approval processes that you may need in order to implement and manage automation effectively in your environment. Organization-wide automation requires coordination and oversight:

- Where multiple users have the ability to create automated tasks, it is possible for them to create duplicative or even conflicting automation. Consider who should be repsonsible for deconflicting and approving automation that will be deployed in your production environment.
- Automation is often used to enrich indicators (i.e., query various third-party APIs to pull in more data related to a node). Some third-party APIs may impose query limits, may be subject to a license or subscription fee, or both. Consider how to balance effective use of automation without overusing or exceeding any applicable quotas.
- Some automation may be used to apply tags to nodes or “push” tags from one node to related nodes - effectively automating the process of making an analytical assertion. Consider carefully under what circumstances this should be automated, and who should review or approve the analysis logic used to make the assertion.

Automation and Error Handling

If the Storm executed by a piece of automation encounters an error condition, the automation will cease execution and exit. For automation that operates over large numbers of nodes or performs a long-running task, an unexpected error can cause the automation to halt in the middle and fail to complete.

In addition to general testing of your Storm code (and Storm logic!), using the *“Try” Operator* when adding or modifying data will ensure your Storm code will “warn and continue” if it encounters bad data vs. generating an error and halting.

We also encourage you to build additional error-checking and error-handling into your automation as appropriate.

Users should keep the *Storm Operating Concepts* in mind when writing automation. Automation frequently operates on nodes or other Synapse data; knowing what nodes are in the Storm pipeline for your automation will help significantly with troubleshooting any issues.

Triggers and Cron

Triggers and cron are similar in terms of how they are implemented and managed.

- **Permissions.** Synapse uses permissions to determine who can create, modify, and delete triggers and cron jobs. These permissions must be explicitly granted to users and/or roles. See the Cortex permissions section of the Synapse Admin Guide for a list of `cron.*` and `trigger.*` permissions.
- **Execution.** Both triggers and cron jobs execute with the permissions of a designated user associated with the individual trigger or cron job. By default, this is the user who creates the trigger or cron job. The user can be changed (e.g., for organizations that wish to use a dedicated account for automation tasks) using the `set()` method of the *trigger* or *cronjob* primitives, respectively.
- **Scope.** Both triggers and cron jobs run **within a specific view**; any changes made to Synapse’s data by the trigger or cron job are made (written) to the topmost (writeable) layer of the view. This view-specific behavior is transparent when using a simple Synapse implementation consisting of a single Cortex with a single layer and a single view (Synapse’s default configuration). Organizations using multiple views or that frequently fork views should consider the impact of your view architecture on automation deployment and behavior.

Note: The use of sensitive information (e.g., credentials, API keys, etc.) in the Storm that is executed by a trigger or cron job is **strongly discouraged**. If your automation needs to make use of any sensitive information, we recommend creating a Power-Up with associated Storm commands. The Power-Up architecture allows users to run commands that make use of sensitive information while ensuring that information is not exposed. See the Rapid Power-Up Development section of the Synapse Developer Guide for details.

Cron

Cron in Synapse is similar to the well-known cron utility. Cron jobs execute their associated Storm on a specified schedule. Jobs can be written to execute once (using the *cron.at* command) or on a recurring basis (using *cron.add*).

Tip: When scheduling cron jobs, Synapse interprets all times as UTC.

Configuration and Management

- **Storage.** Cron jobs are stored **globally** (within the Cortex). When **viewing** cron jobs, (e.g., with the *cron.list* command), Synapse returns all cron jobs in the Cortex, regardless of the view the `cron.list` command is executed in.
- **Execution.** Cron jobs must be assigned to a specific **view** where they execute. By default, this is the view where the cron job is created. If the view that a cron job runs in is deleted, the cron job **remains** (within the Cortex) but is effectively orphaned until it is assigned to a new view (i.e., using the *$lib.cron.move(prefix, view)* library) or deleted if no longer needed.
- **Permissions.** Cron jobs execute with the privileges of a designated user (by default, the user who creates the cron job). We strongly encourage the use of least privilege; the cron job's account should have the permissions required to execute the associated Storm, but no more. One option is for organizations to create a dedicated account for use with automation in Synapse.

  The owner (creator) of a cron job can be modified using the Storm *$lib.cron.get(prefix)* library and the `set()` method of the *cronjob* primitive. For example:

  ```
  $mycron=$lib.cron.get(<cron_iden>) $mycron.set(creator, <new_creator_iden>)
  ```

  Users (and roles) cannot create or manage cron jobs by default. The various `cron.*` permissions must be granted to users or roles that should be allowed to work with cron jobs in your environment. See the Cortex permissions section of the Synapse Admin Guide for details on the `cron.*` permissions and associated gates.

  Note: Where a user has **admin** privileges, all permissions checks are bypassed. This means that users can create and manage cron jobs in views that they fork.

- **Managing Cron Jobs.** Cron jobs can be created, viewed, and managed using the various Storm *cron* commands, the *cronjob* primitive, or the *$lib.cron* libraries. In *Optic*, cron jobs can also be created and managed using the *Admin Tool*.

Use Cases

Because cron jobs are scheduled, they are most appropriate for automating routine tasks, non-urgent tasks, or resource-intensive tasks that should be scheduled to minimize impact on operations.

"What" a cron job does is limited only by your imagination (and your Storm skills). Examples of common cron use cases include:

- **Data ingest.** Periodically ingest / synchronize data that you want to load into Synapse on a regular basis.
- **Housekeeping.** Perform one-time or periodic "maintenance" tasks. These may include a one-time sweep to "backfill" missing data (like IP geolocation data) or a periodic sweep to check and set missing properties (such as tag definitions).
- **Process intensive jobs.** Data enrichment may be resource intensive if it generates a significant number of write operations. If you regularly perform routine (non-urgent) enrichment, it can be scheduled to run when it will have less impact on users.
- **Periodic hunting.** New data and annotations (tags) are continually added to Synapse. Encoding "hunt" logic in Storm and periodically running hunts can help ensure you are continually reviewing new data for updated indicators or activity.

Tip: A cron job can use Storm to call a macro, which may provide greater flexibility in storing and managing the Storm code executed by the cron job.

Syntax

In Storm, cron jobs are primarily created and managed using the Storm *cron* commands.

Once a cron job is created, you can modify many of its properties (such as its name and description, or the Storm associated with the job). However, you cannot modify other aspects of the job, such as its schedule. To change those conditions, you must disable (or delete) and re-create the cron job.

Variables

Every cron job has an associated Storm variable `$auto` that is automatically populated when the job runs. The `$auto` variable is a dictionary which contains the following keys:

`$auto.iden`
: The identifier of the cron job.

`$auto.type`
: The type of automation. For a cron job this value will be `cron`.

These job-specific variables can be referenced by the Storm code executed by the cron job. See *Storm Reference - Advanced - Variables* for information on the use of variables in Storm.

Examples

The examples below illustrate using the Storm *cron* commands (`cron.at`, `cron.add`) to create new cron jobs. See the help for each command for options and additional details.

One-time (non-recurring) cron job example

You want to use a one-time cron job that runs during off hours to perform some data cleanup. Specifically, you want a job that will lift all existing `media:news` nodes and remove (delete) the deprecated `:author` property at 0200 UTC.

```
storm> cron.at --hour 2 { media:news:author [ -:author ] }
Created cron job: d00e07288ff15ae2c09cf51b67ec18eb
```

We can view the details of this cron job using the `cron.list` command:

```
storm> cron.list
user      iden       view       en? rpt? now? err? # start last start       last end     ␣
↪     query
root      d00e0728.. ce7206f2.. Y   N    N          0       Never             Never        ␣
↪     media:news:author [ -:author ]
```

The output of `cron.list` includes the following columns:

- **user** - the user that the job runs as (typically the user who created the job).
- **iden** - the first eight characters of the cron job's identifier (iden).
- **view** - the first eight characters of the iden of the view the job executes in. For "orphaned" cron jobs, this will be the job's last view (before it was orphaned).
- **en?** - whether the job is currently enabled or disabled (Y/N).
- **rpt?** - whether the job is scheduled to repeat (Y/N).
- **now?** - whether the job is currently executing (Y/N).
- **err?** - whether the last job execution encountered an error (X or empty).
- **# start** - the number of times the job has started / executed (since the last restart of Synapse).
- **last start** - the date and time the job last started (or attempted to start).
- **last end** - the date and time the job last finished (or exited).
- **query** - the Storm query executed by the cron job.

Tip: Detailed information about individual cron jobs can be retrieved with the *cron.stat* command.

Hourly cron job example

You want to create a cron job that will run every hour to download the latest MISP data using the `misp.sync` command.

```
cron.add --hour +1 { misp.sync }
```

Tip: The `misp.sync` command is added by the Synapse-MISP Power-Up.

Custom schedule example

Because Synapse is continually updated with new data, you want to periodically check for any new files of interest that have not yet been identified by an analyst. Specifically, you want to look for files (`file:bytes` nodes) that make DNS queries (`inet:dns:request`) for "known bad" FQDNs (e.g., tagged `#cno.mal`) where the files have not yet been tagged. You want to run the job three times a week (every Tuesday, Thursday, and Saturday at 2000 UTC) and tag the files for review (`#int.review.malware`).

```
cron.add --day Tue,Thu,Sat --hour 20 { inet:fqdn#cno.mal -> inet:dns:request ->
file:bytes -#cno.mal [ +#int.review.malware ] }
```

Triggers

Triggers are "event-driven" automation. As their name implies, they trigger ("fire") their associated Storm when specific events occur in Synapse's data store. Triggers can fire on the following events:

- Adding a node (`node:add`)
- Deleting a node (`node:del`)
- Setting (or modifying) a property (`prop:set`)
- Adding a tag to a node (`tag:add`)
- Deleting a tag from a node (`tag:del`)
- Adding a light edge (`edge:add`)
- Deleting a light edge (`edge:del`).

When creating a trigger, you must specify the type of event the trigger should fire on. In addition, each event requires a specific object (a form, property, tag, or edge) to act upon. For example, if you write a trigger to fire on a `node:add` event, you must specify the type of node (form) associated with the event; you cannot create a `node:add` trigger that fires on "any / all nodes". For `tag:*` and `edge:*` triggers, you have the option to limit the trigger to tags or edges associated with specific forms, or have the trigger apply to any / all forms.

Note: The node(s) that cause a trigger to fire are considered **inbound** to the Storm code executed by the trigger.

Proper trigger execution may depend on the timing and order of events with respect to creating nodes, setting properties, and so on. For example, you may write a trigger based on a `node:add` action that fails to perform as expected because you actually need the trigger to fire on a `prop:set` operation for that node. As always, we recommend that you test triggers (and any other automation) before putting them into production.

Configuration and Management

- **Storage.** Triggers are stored within a **view**. If the view that a trigger resides in is deleted, **the trigger is also deleted**. Triggers can be moved to another view using the `move()` method of the *trigger* primitive. (In *Optic*, when using the View Task Bar to merge a view, you are prompted to also merge any triggers in the view.)

  Because triggers are stored within individual views, when **viewing** triggers (e.g., with the *trigger.list* command), Synapse returns only those triggers within the current view.

- **Execution.** Triggers "fire" when a change occurs within Synapse - that is, when a **write** operation occurs. As triggers reside (and execute) within a **view**, they fire on changes to the view's writable (topmost) **layer.** A trigger will **not** fire on changes made in an underlying layer (i.e., of a parent view).

Similarly, a trigger that resides in a **parent** view will **not** fire on changes made in a fork of that view. However, when data from the forked view is merged into the parent view, merging (writing) the data will cause any relevant triggers to fire.

Triggers fire immediately when their associated event occurs; but they **only** execute when that event occurs. If the Storm executed by the trigger depends on a resource (process, service, etc.) that is not available when the trigger fires, execution will simply fail; Synapse will not "try again" later. (Whether the trigger fails silently or by logging an error will depend on the Storm and / or any additional resources used or called by the trigger.)

Similarly, triggers do not operate "retroactively" on existing data. If you write a new trigger to fire when the tag `my.tag` is applied to a `hash:md5` node, the trigger will have no effect on existing `hash:md5` nodes that already have the tag.

- **Permissions.** Triggers execute with the privileges of a designated user (by default, the user who creates the trigger). We strongly encourage the use of least privilege; the trigger's account should have the permissions required to execute the associated Storm, but no more. One option is for organizations to create a dedicated account for use with automation in Synapse.

  The owner (user) of a trigger can be modified using the Storm *$lib.trigger.get(iden)* library and the `set()` method of the *trigger* primitive. For example:

  ```
  $mytrigger=$lib.trigger.get(<trigger_iden>) $mytrigger.set(user, <new_user_iden>)
  ```

  Triggers run as a specified user, but they execute based on particular changes to data in Synapse. This means that a lower-privileged user could make a change (such as creating a node) that causes a higher-privileged trigger to execute and perform actions that the user would not be able to do themselves.

  Users (and roles) cannot create or manage triggers by default. The various `trigger.*` permissions must be granted to users or roles that should be allowed to work with triggers in your environment. See the Cortex permissions section of the Synapse Admin Guide for details on the `trigger.*` permissions and associated gates.

  Note: Where a user has **admin** privileges, all permissions checks are bypassed. This means that users can create and manage triggers in views that they fork.

- **Managing Triggers.** Triggers can be created, viewed, and managed using the various Storm *trigger* commands, the *trigger* primitive, or the *$lib.trigger* libraries. In *Optic*, triggers can be created and managed using the *Admin Tool* or the VIEWS tab of the *Workspaces Tool*.

Use Cases

Triggers are "event driven" and execute their Storm **immediately** when their associated event (change) occurs. As such, triggers are most appropriate for automating tasks that should occur right away (e.g., based on efficiency or importance). Example use cases for triggers include:

- **Performing enrichment.** There may be circumstances where you **always** want to retrieve additional information about an object (node) within Synapse. For example, whenever a unicast IPv4 (`inet:ipv4` node) is added to Synapse, you want to automatically look up Autonomous System (AS), geolocation, and network whois data. You can use a trigger to enrich the `inet:ipv4` using the appropriate Power-Ups as soon as the IPv4 is created (e.g., by firing on a `node:add` event).

  Similarly, when a node is assessed to be malicious (e.g., associated with a threat cluster or malware family, and tagged appropriately), you may wish to immediately collect additional information about that node. A trigger that fires on a `tag:add` event could be used to call multiple Power-Ups using a more extensive "enrichment" query (or macro).

- **Encoding analytical logic.** You can use Storm to encode your analysis logic, such as the criteria or decision process used to apply a tag. As a simplified example, assume you have identified an IPv4 address as a DNS sinkhole. You assess that any FQDN resolving to the IPv4 is highly likely to be a sinkholed domain. When a DNS A node (`inet:dns:a`) is created where the associated IPv4 (`:ipv4` property) is the IP of the sinkhole (a `prop:set` event), a trigger can automatically tag the associated FQDN as sinkholed. If you want an analyst to confirm the assessment (vs. applying it in a fully automated fashion), you can apply a "review" tag instead. Alternatively, if additional criteria would better support your assessment (e.g., if the DNS A resolution occurred during a particular time window when the sinkhole IP was known to be in use, or if the resolution was based on a live/active DNS lookup vs. a passively observed response), you could modify your trigger's Storm to reflect these additional "checks" or criteria.

  You can similarly encode logic in Storm to support retrohunting, threat clustering, or other workflows.

- **Automating repetitive tasks.** Any process that analysts identify as repetitive may benefit from automation. Analysts may identify cases where, when they perform a specific action in Synapse, they always perform several additional actions. For example, when an analyst tags a particular node (such as a `file:bytes` node), they always want to apply the same tag to a set of "related" nodes (such as the `hash:md5`, `hash:sha1`, etc. that represent the file's hashes). Similarly, if a `file:bytes` node queries a "known bad" FQDN (via an `inet:dns:request` node), analysts also want to apply the tag from the FQDN to both the DNS request and the file. Using a trigger (on a `tag:add` event) saves manual work by the analyst and ensures the additional tags are applied consistently.

Tip: A trigger can use Storm to call a macro, which may provide greater flexibility in storing and managing the Storm code executed by the trigger.

Syntax

In Storm, triggers are created, modified, viewed, enabled, disabled, and deleted using the Storm *trigger* commands. In *Optic*, triggers can also be managed through the *Admin Tool* or the *Workspaces Tool*.

Once a trigger is created, you can modify many of its properties (such as its name and description, or the Storm associated with the trigger). However, you cannot modify the conditions that cause the trigger to fire. To change those conditions, you must disable (or delete) and re-create the trigger.

Variables

Every trigger has an associated Storm variable `$auto` that is automatically populated when the trigger executes. The `$auto` variable is a dictionary which contains the following keys:

`$auto.iden`
: The identifier of the Trigger.

`$auto.type`
: The type of automation. For a trigger this value will be `trigger`.

`$auto.opts`
: Dictionary containing trigger-specific runtime information. This includes the following keys:

  `$auto.opts.form`
  : The form of the triggering node.

  `$auto.opts.propfull`
  : The full name of the property that was set on the node. Only present on `prop:set` triggers.

$auto.opts.propname
The relative name of the property that was set on the node. Does not include a leading `:`. Only present on `prop:set` triggers.

$auto.opts.tag
The tag which caused the trigger to fire. Only present on `tag:add` and `tag:del` triggers.

$auto.opts.valu
The value of the triggering node.

$auto.opts.verb
The name of the light edge. Only present on `edge:add` and `edge:del` triggers.

$auto.opts.n2iden
The iden of the node on the other end of the edge. Only present on `edge:add` and `edge:del` triggers.

These trigger-specific variables can be referenced by the Storm code executed by the trigger. See *Storm Reference - Advanced - Variables* for information on the use of variables in Storm.

Examples

The examples below illustrate using the Storm *trigger.add* command to create new triggers. See the command help for options and additional details.

Recall from the *Storm Operating Concepts* that Storm operations are "chained" together to act as a pipeline through which nodes are "sent" and acted upon. Regardless of the type of trigger (`prop:set`, `tag:add`), the **node** associated with the trigger event is sent **inbound** to the Storm executed by the trigger.

Tip: By default, the Storm code executed by a trigger runs **inline**. In other words, when a process (typically a Storm operation or Storm query) causes a trigger to fire, the Storm associated with the trigger will run **immediately and in full**. Conceptually, it is as though all of the trigger's Storm code and any additional Storm that the trigger calls (such as a macro) or causes to run (such as another trigger) are inserted into the middle of the original Storm query that fired the trigger, and executed as part of that query.

This inline execution can impact the original query's performance, depending on the Storm executed by the trigger and the number of nodes causing the trigger to fire. Specifically, if a trigger (and associated Storm) is fired based on an interactive Storm query from a user, Synapse's interface (the Storm CLI or the Optic UI) will "block" until the Storm executes in full.

The `--async` option can be used when creating a trigger to specify that the trigger should run **in the background** as opposed to inline. This will cause the trigger event to be stored in a persistent queue, which will then be consumed automatically by the Cortex.

Executing the trigger asynchronously means that changes made by the trigger (e.g., such as data enrichment) will not be available to the user until the associated Storm finishes running. However, the user can continue working in the meantime because the Synapse interface will not "block".

To change whether or not an **existing** trigger runs asynchronously, use the Storm *$lib.trigger.get(iden)* library and the `set()` method of the *trigger* primitive. For example:

```
$mytrigger=$lib.trigger.get(<trigger_iden>) $mytrigger.set(async, $lib.true)
```

Or:

```
$mytrigger=$lib.trigger.get(<trigger_iden>) $mytrigger.set(async, $lib.false)
```

prop:set example

You have identified a handful of email addresses that are consistently used by various sinkhole organizations to register their sinkholed FQDNs, and tagged the email addresses `#cno.infra.dns.sink.hole` to show they are associated with sinkhole infrastructure. You want to add a trigger that will fire when any of these email addresses is set as the `:email` property of an `inet:whois:contact` node and tag the associated FQDN as sinkholed (`#cno.infra.dns.sink.holed`). Because the trigger is a simple one, you want it to run inline (i.e., you do not need the `--async` option).

```
storm> trigger.add prop:set --name 'Tag sinkholed FQDNs based on known sinkholer email' -
↪-prop inet:whois:contact:email --query { +{ :email-> inet:email +#cno.infra.dns.sink.
↪hole } -> inet:whois:rec -> inet:fqdn [ +#cno.infra.dns.sink.holed ] }
Added trigger: c4f859cb0b43fb2b90ce2673ca313b44
```

Tip: For `prop:set` and `prop:del` triggers, the node whose property caused the trigger to fire is sent inbound to the trigger's Storm.

We can view the newly created trigger using `trigger.list`:

```
storm> trigger.list
user       iden                             view                             en?   
↪async? cond      object                     storm query
root       c4f859cb0b43fb2b90ce2673ca313b44 ce7206f21840c82b41b28f0524603a6b true  
↪false   prop:set  inet:whois:contact:email            +{ :email-> inet:email +#cno.
↪infra.dns.sink.hole } -> inet:whois:rec -> inet:fqdn [ +#cno.infra.dns.sink.holed ]
```

The output of `trigger.list` includes the following columns:

- **user** - the user that the trigger runs as (typically the user who created the trigger).
- **iden** - the trigger's identifier (iden).
- **view** - the iden of the view in which the trigger resides / runs.
- **en?** - whether the trigger is currently enabled or disabled.
- **async?** - whether the trigger will run asynchronously / in the background.
- **cond** - the condition that causes the trigger to fire.
- **object** - the object the trigger operates on (in this case, the `:email` property of an `inet:whois:contact` node). May reference multiple objects (e.g., if a `tag:add` trigger only fires when the tag is added to a specific form, both the tag and form will be listed).
- **storm query** - the Storm to be executed when the trigger fires.

node:add example

Whenever an IPv4 node is added to Synapse, you want to immediately retrieve the associated Autonomous System (AS), geolocation, and DNS PTR data using various Storm commands.

```
storm> trigger.add node:add --name 'Basic IPv4 enrichment' --form inet:ipv4 --query {
↪maxmind | nettools.dns }
Added trigger: dcc73dcda930ec89715266c2c6668a10
```

Tip: For `node:add` and `node:del` triggers, the node that caused the trigger to fire is sent inbound to the trigger's Storm.

The maxmind and nettools.dns commands are added to Synapse by the Synapse-Maxmind and Synapse-Nettools Power-Ups, respectively.

edge:add example

You are using risk:attack nodes in Synapse to represent attacks, and linking the attack to key objects used in the attack with a -(uses)> light edge. (For example, a risk:attack representing a phishing attempt can be linked to the relevant email message (file:bytes node) or email metadata nodes (such as inet:email:message, inet:email:message:attachment, or inet:email:message:link).

If the object "used" in the attack has an associated TTP tag (such as #cno.ttp.phish.attach to represent a malicious attachment used in a phishing attack), you want to pivot to the technique (ou:technique) represented by the tag (e.g., "Spear phishing attachment"), and create another uses edge to show the attack uses the specified technique.

```
storm> trigger.add edge:add --name 'Link technique with attack'--verb uses --form␣
↪risk:attack --query { $attack=$node { -(uses)> * +$auto.opts.n2iden -> # ->␣
↪ou:technique:tag [ <(uses)+ { yield $attack } ] } }
Added trigger: 4dd4851aefd174271fb8e9699ffc3813
```

Tip: Recall that edges have a direction, regardless of whether the edge relationship is specified "left to right" or "right to left" in Storm (e.g., n1 -(uses)> n2 or n2 <(uses)- n1).

For edge:add and edge:del triggers, the n1 node (source node) for the edge that caused the trigger to fire is sent inbound to the trigger's Storm.

The Storm for this trigger is broken down below (with comments) for clarity:

```
// Set the variable $attack to the risk:attack node for later use
$attack=$node

// Use a subquery to traverse all 'uses' edges to any / all forms
{
   -(uses)> *

   // Filter to the n2 node that was just linked, as referenced by the
   // trigger-specific $auto.opts.n2iden variable
   +$auto.opts.n2iden

   // Pivot to the syn:tag nodes associated with any tags on n2
   -> #

   // Pivot to any techniques associated with any tags
   -> ou:technique:tag

   // Create a 'uses' edge between the technique(s) and the risk:attack node
   [ <(uses)+ { yield $attack } ]
}
```

tag:add example

When you associate a file with a malware family, you record that assessment on the file:bytes node with a tag (e.g., #cno.mal.redtree to represent the 'redtree' malware family). When you tag the file, you want to automatically copy the tag to the file's MD5, SHA1, SHA256, and SHA512 hashes for situational awareness.

```
storm> trigger.add tag:add --name 'Push malware tags from file to hashes' --form␣
↪file:bytes --tag cno.mal.** --query { tee { :md5 -> hash:md5 } { :sha1 -> hash:sha1 }
↪{ :sha256 -> hash:sha256 } { :sha512 -> hash:sha512 } | [ +#$auto.opts.tag ] }
Added trigger: daccd0954a78ace8a7b05e53206b8305
```

Tip: For `tag:add` and `tag:del` triggers, the node that was tagged (or untagged) to cause the trigger to fire is sent inbound to the trigger's Storm.

The trigger above uses the trigger-specific `$auto.opts.tag` variable to reference the tag that caused the trigger to fire.

tag:del example

Similar to the example above, when your assessment changes and you want to **remove** a tag from a file (`file:bytes` node), you want to automatically remove that same tag from the file's associated hashes.

```
storm> trigger.add tag:add --name 'Untag hashes when untagging a file' --form file:bytes␣
↪--tag cno.mal.** --query { tee { :md5 -> hash:md5 } { :sha1 -> hash:sha1 } { :sha256 ->
↪ hash:sha256 } { :sha512 -> hash:sha512 } | [ -#$auto.opts.tag ] }
Added trigger: 3f47a2b958ccaade261c110bb7cc5bdf
```

Tip: The trigger above uses the trigger-specific `$auto.opts.tag` variable to reference the tag that caused the trigger to fire.

Macros

A macro is a stored Storm query / set of Storm code that can be executed on demand. Strictly speaking, macros are not automation - they do not execute on their own. However, they do provide a means to "automate" tasks by allowing an analyst to easily and consistently run Storm that would otherwise need to be manually entered as a query. Anything that assists an analyst's workflow - from executing a commonly used pivot to performing a detailed set of data enrichment and operations - is a candidate for a macro if it simplifies analysts' work.

Tip: In addition to macros, *Optic* includes features such as Node Actions and Bookmarks that can be used to save and execute Storm on demand.

Macros are also commonly used with (called by) triggers or cron jobs. In particular, where a trigger or cron job executes longer or more detailed Storm, it may be easier to store and maintain that Storm within a macro instead of within the trigger or cron job itself.

As a stored Storm query, a macro is meant to operate on nodes. You must either specify (lift) the nodes to be operated on within the macro itself, or provide the nodes as input to the macro (i.e., by sending the results of an existing Storm query to the *macro.exec* command that runs the macro).

A macro will error if it receives nodes that cannot be processed by the associated Storm code. A "best practice" when writing macros that take nodes as input is to include a Storm filter operation within the macro to allow only those nodes that the macro expects and can operate on; any nodes not allowed by the filter are dropped from the Storm pipeline.

Similarly, if you execute additional Storm inline after the macro runs, that Storm must be appropriate for any nodes that exit the macro.

Note: The use of sensitive information (e.g., credentials, API keys, etc.) in the Storm that is executed by a macro is

strongly discouraged. If your automation needs to make use of any sensitive information, we recommend creating a Power-Up with associated Storm commands. The Power-Up architecture allows users to run commands that make use of sensitive information while ensuring that information is not exposed. See the Rapid Power-Up Development section of the Synapse Developer Guide for details.

Configuration and Management

- **Storage.** Macros are stored within (global to) a Cortex. Macros are differentiated by **name** (as opposed to triggers and cron jobs, which are differentiated by a unique identifier (iden)). You can change the name of a macros using the *$lib.macro.mod(name, info)* library:

  ```
  $lib.macro.mod('my old poorly chosen macro name',({'name': 'new.name'}))
  ```

  Attempting to rename a macro to a name that already exists will result in a `DupName` error.

- **Execution.** Macros are not restricted to a particular view; they can be called from and executed within any view. Any changes made by the macro's Storm code will be made to the topmost / writeable layer of the view in which the macro is executed.

- **Permissions.** Macros execute with the privileges of the **user who calls the macro**. If a macro is called by a trigger or cron job, the macro executes with the privileges of the associated trigger or cron user.

  By default:

  – Any user can create a macro; you do not need to explicitly grant permissions for users to create them.

  – The user who creates a macro is the owner / admin of the macro, and is the only user who can edit, delete, or modify permissions on the macro.

  – All users can see (read) and execute any macro. If a user attempts to execute a macro that performs actions for which the user does not have permissions, the macro will fail with an `AuthDeny` error.

  Macro permissions can be modified / managed by using the *$lib.macro.grant(name, scope, iden, level)* library. Macros support Synapse's easy permissions convention for setting simplified permissions on objects. You must be **admin** of a macro to modify its permissions. See the *Examples* section below for examples of modifying macro permissions.

- **Managing Macros.** Macros can be created, viewed, and managed using the various Storm *macro* commands or the *$lib.macro* libraries. In *Optic*, macros can be created and edited using the *Storm Editor*.

Use Cases

Macros are a convenient way to save and run frequently used Storm without having to create or type that Storm each time. The Storm can be as simple or advanced as you like.

- **Organizational use.** Macros can be developed for use across entire teams or organizations to support common tasks or workflows such as enrichment or threat hunting. Using a macro makes it easier to perform the task (by calling it with a single Storm command) and also ensures that the task is performed consistently (i.e., in the same way each time) by each user.

- **Personal use.** Users can create macros to store frequently-used or lengthy Storm queries specific to their personal workflow that can be executed easily on demand.

- **Automation.** For triggers or cron jobs that execute longer Storm queries, saving the Storm in a macro may make it easier to set, view, edit, and manage vs. storing the Storm directly as part of the trigger or cron job.

- **Flexibility.** Because macros are composed in Storm and executed via a Storm command, they can be executed any way Storm can be executed (e.g., on demand or called as part of a trigger or cron job). Macros are ideal for Storm that performs a task or set of tasks that you may want to execute in a variety of ways.

Syntax

In Storm, macros are created, modified, viewed, and deleted using the Storm *macro* commands. In *Optic*, macros can also be managed through the *Storm Editor*.

Permissions for macros are managed using the *$lib.macro.grant(name, scope, iden, level)* library.

Examples

As "stored Storm", macros can contain any Storm that simplifies your analysis workflow - from the very simple to longer, more detailed queries. The command `macro.exec <macro_name>` is much simpler to type than Storm that you have to remember or recreate each time. The examples below are relatively simple for illustrative purposes; that said, any Storm that you want to easily and consistently run on a regular basis, no matter how "simple" or "complex", is a candidate for a macro (or potentially for other automation).

The examples below show creating macros using the `macro.set` command.

Self-contained macro - example

You have a set of IPv4 addresses that you have identified as sinkholes (tagged `#cno.infra.dns.sink.hole`). By analyzing associated DNS and domain whois data, you have tagged several FQDNs resolving to the IPv4s as sinkholed domains (`#cno.infra.dns.sink.holed`). You want to periodically check the sinkhole IPv4s for newly sinkholed FQDNs. To do this, you create a macro called `sinkhole.check` that will:

- query a passive DNS data source to check for new FQDNs resolving to the various sinkhole IPv4s;
- perform a live DNS A lookup on the FQDNs to double-check against the passive DNS results;
- retrieve the domains' current whois records; and
- tag the FQDNs for review, including a timestamp so the reviewer knows when the FQDN was added to the set of domains for review.

```
storm> macro.set sinkhole.check { $now=$lib.time.now() inet:ipv4#cno.infra.dns.sink.hole␣
↪| alienvault.otx.pdns --yield | -> inet:fqdn :zone -> inet:fqdn -#cno.infra.dns.sink.
↪holed | uniq | nettools.dns | nettools.whois | [ +#int.review.sinkhole=$now ] }
Set macro: sinkhole.check
```

The Storm for this macro is broken down below (with comments) for clarity. (You can include comments within a macro; Synapse will ignore comment lines during execution.)

```
// Get the current time (in UTC) to use for the tag timestamp
$now=$lib.time.now()

// Lift IPv4 nodes tagged as sinkholes
inet:ipv4#cno.infra.dns.sink.hole

// Obtain PDNS information from AlienVault and yield the resulting inet:dns:a nodes
alienvault.otx.pdns --yield |

// Pivot to the FQDNs and then to the FQDN zones
-> inet:fqdn :zone -> inet:fqdn
```

(continues on next page)

(continued from previous page)

```
// Filter out FQDNs that have already been identified as sinkholed
-#cno.infra.dns.sink.holed |

// De-duplicate results
uniq |

// Obtain the current / live DNS A and whois data for the FQDNs
nettools.dns | nettools.whois |

// Tag the FQDNs for review and set the current timestamp
[ +#int.review.sinkhole=$now ]
```

An analyst can now lift the FQDNs that need to be reviewed using the `#int.review.sinkhole` tag and review the associated data. If the FQDN is confirmed to be sinkholed, the analyst can apply the `#cno.infra.dns.sink.holed` tag and remove the `#int` tag once the FQDN has been reviewed.

If additional criteria would allow you to confirm that the FQDNs were sinkholed (e.g., some additional information in the whois data, or use of a particular DNS name server / DNS NS record), you could encode this logic using Storm and simply tag the relevant FQDNs as sinkholed vs. requiring analyst review.

Tip: This macro is "self-contained" (does not expect inbound ndoes); the macro's Storm lifts the nodes that the macro operates on. (This does not prevent a user from "sending" nodes to the macro - which may cause unexpected effects - but as written, the macro can execute on its own.)

This macro can be executed "on demand" with the `macro.exec` command, or could be configured to run automatically as a cron job.

The `alienvault.otx.pdns` command is installed by the Synapse-AlienVault Power-Up. The `nettools.*` commands are installed by the Synapse-Nettools Power-Up.

Macro that takes nodes as input - example

One of Synapse's strengths is the ability to retrieve data from a wide range of external sources, typically by using a Storm command provided by a Power-Up. The Storm commands allow an analyst to ingest data into Synpase and represent disparate data in a consistent manner for review and analysis. An analyst who wants to investigate an object (such as an indicator) in Synapse will commonly run multiple Storm commands to retrieve data from any / all data sources (a process commonly known as "enrichment").

Instead of requiring the analyst to remember and run multiple individual Storm commands for enrichment, you want to create a macro called `enrich` that can take a variety of different nodes as input, and automatically run the appropriate Storm commands to call the data sources that can enrich the input node(s).

Once again, we create the macro with the `macro.set` command. Because of the length of the associated Storm, the full <`storm_query`> is provided below for readability.

```
macro.set enrich { <storm_query> }
```

The content of the macro (the <`storm_query`>, with comments):

```
// Filter inbound nodes to supported forms only
+(hash:md5 or hash:sha1 or hash:sha256 or inet:fqdn or inet:ipv4)

// Switch statement to handle different forms
```

(continues on next page)

(continued from previous page)

```
switch $node.form() {

  "hash:md5": {
      { | virustotal.file.report | virustotal.file.behavior }
  }

  "hash:sha1": {
      { | virustotal.file.report | virustotal.file.behavior }
  }

  "hash:sha256": {
      { | virustotal.file.report | virustotal.file.behavior | alienvault.otx.files }
  }

  "inet:fqdn": {

      // For FQDN zones
      {
          +:iszone=1 +:issuffix=0
          { | nettools.dns --type A AAAA CNAME MX NS SOA TXT | nettools.whois |␣
↪virustotal.pdns | virustotal.commfiles | alienvault.otx.domain | alienvault.otx.pdns }
      }

      // For FQDN subdomains
      {
          +:iszone=0 +:issuffix=0
          { | nettools.dns --type A AAAA CNAME | virustotal.pdns | virustotal.commfiles␣
↪| alienvault.otx.domain | alienvault.otx.pdns }
      }
  }

  "inet:ipv4": {

      +:type=unicast
      { | maxmind | nettools.dns | nettools.whois | virustotal.pdns | virustotal.
↪commfiles | censys.hosts.enrich | alienvault.otx.ip | alienvault.otx.pdns }
  }

  // Default case for any forms not specified above
  *: {
      // Do nothing
  }
}
```

Instead of running multiple Storm commands, an analyst can now run a single macro that will perform enrichment by "intelligently" querying all available data sources that can provide information about a particular kind of indicator.

Tip: This macro requires nodes as input; the macro's Storm does not lift any nodes to operate on. The initial filter statement is provided as basic error checking to ensure only nodes that the macro knows how to process are sent to the remaining macro code. (Technically, the final "default" switch option (`*: { }`) will handle any unknown forms; using the filter simply "drops" the nodes earlier in the pipeline. If you **want** any unhandled forms to pass through the macro (so they are available for additional Storm operations after the macro finishes), the filter should be removed.)

This macro can be executed "on demand" with the `macro.exec` command. If some condition should cause enrichment to occur automatically (e.g., applying a particular tag to a node), the macro could be configured to run as a trigger.

The example commands used in the macro are all installed by various Synapse Power-Ups.

The macro makes use of a *Switch Statement* (part of Storm's control flow features) to handle different kinds of nodes (forms).

Macros run inline by default; when a user calls a macro, the full macro code executes to completion. For enrichment-type macros, depending on the number of data sources queried and nodes created, enrichment can take some time to run, which can cause the Synapse interface (the Storm CLI or the Optic UI) to "block" for the user until the Storm executes in full. The macro code can be run asynchronously using the Storm *background* command. This "unblocks" the CLI / UI, but the full set of results from enrichment will not be available until the macro finishes executing.

Modify macro permissions - examples

You can modify permissions on a macro using the *$lib.macro.grant(name, scope, iden, level)* library to grant (or revoke) access. The *$lib.auth.users.byname(name)* and *$lib.auth.roles.byname(name)* libraries can be used to obtain a user or role object and retrieve the associated identifier (iden).

Make user "ron the cat" co-admin of the "sinkhole.check" macro:

```
storm> $user=$lib.auth.users.byname('ron the cat') $iden=$user.iden $lib.macro.
↪grant(sinkhole.check,users,$iden,3)
```

Allow the "cattribution analysts" role to edit the "enrich" macro:

```
storm> $role=$lib.auth.roles.byname('cattribution analysts') $iden=$role.iden $lib.macro.
↪grant(enrich,roles,$iden,2)
```

Deny the "interns" role access to the "enrich" macro:

```
storm> $role=$lib.auth.roles.byname(interns) $iden=$role.iden $lib.macro.grant(enrich,
↪roles,$iden,0)
```

Tip: Easy permissions allow you to assign common permissions based on a corresponding integer value:

- 0 - Deny
- 1 - Read (including execute)
- 2 - Edit (modify content; includes Read)
- 3 - Admin (delete the object and modify its permissions; includes Edit)

Dmons

A *Dmon* is a long-running or recurring query or process that runs continuously in the background, similar to a traditional Linux or Unix daemon.

Variables

Dmons will have the storm variable `$auto` populated when they run. The `$auto` variable is a dictionary which contains the following keys:

`$auto.iden`
The identifier of the Dmon.

`$auto.type`
The type of automation. For a Dmon this value will be `dmon`.

Note: If the variable `$auto` was captured during the creation of the Dmon, the variable will **not** be mapped in.

Syntax

Users can interact with dmons using the Storm *dmon* commands and the *$lib.dmon* Storm libraries.

3.7 Storm Advanced

The are several more advanced Storm language language concepts which are documented in the following sections.

3.7.1 Storm Reference - Advanced - Variables

Storm supports the use of **variables.** A *Variable* is a value that can change depending on conditions or on information passed to the Storm query. (Contrast this with a *Constant*, which is a value that is fixed and does not change.)

Variables can be used in a variety of ways, such as providing more efficient ways to reference node properties; facilitating bulk operations; or writing extensions to Synapse (such as Power-Ups) in Storm.

These documents approach variables and their use from a **user** standpoint and aim to provide sufficient background for users to understand and begin to use variables. They do not provide an in-depth discussion of variables and their use. See the *Synapse Developer Guide* for more developer-focused topics.

Note: It is important to keep the high-level *Storm Operating Concepts* in mind when writing Storm queries or code. This is especially true when working with variables, control flow, and other more advanced concepts.

Variable Concepts

Variable Scope

A variable's **scope** is its lifetime and under what conditions it may be accessed. There are two dimensions that impact a variable's scope: its **call frame** and its **runtime safety** ("runtsafety").

Call Frame

A variable's **call frame** is where the variable is used. The main Storm query starts with its own call frame, and each call to a "pure" Storm command, function, or subquery creates a new call frame. The new call frame gets a copy of all the variables from the calling call frame. Changes to existing variables or the creation of new variables within the new call frame do not impact the calling scope.

Runtsafe vs. Non-Runtsafe

An important distinction to keep in mind when using variables in Storm is whether the variable is runtime-safe (*Runtsafe*) or non-runtime safe (*Non-Runtsafe*).

A variable that is **runtsafe** has a value independent of any nodes passing through the Storm pipeline. For example, a variable whose value is explicitly set, such as `$string = mystring` or `$ipv4 = 8.8.8.8` is considered runtsafe because the value does not change / is not affected by the specific node passing through the Storm pipeline (i.e., by the Storm runtime).

A variable that is **non-runtsafe** has a value derived from a node passing through the Storm pipeline. For example, a variable whose value is set to a node property value may change based on the specific node passing through the Storm pipeline at the time. In other words, if your Storm query is operating on a set of DNS A nodes (`inet:dns:a`) and you define the variable `$fqdn = :fqdn` to set the variable to the value of the `:fqdn` property, the value of the variable will change based on the value of that property for the current `inet:dns:a` node.

All non-runtsafe variables are **scoped** to an individual node as it passes through the Storm pipeline. This means that a variable's value based on a given node is not available when processing a different node (at least not without using special commands, methods, or libraries). In other words, the path of a particular node as it passes through the Storm pipeline is its own scope.

Note: The "safe" in non-runtsafe should **not** be interpreted to mean that the use of non-runtsafe variables is somehow "risky" or involves insecure programming or processing of data. It simply means the value of the variable is not safe from changing (i.e., it may change) as the Storm pipeline progresses.

Types of Variables

Storm supports two types of variables:

- **Built-in variables.** Built-in variables facilitate many common Storm operations. They may vary in their scope and in the context in which they can be used.
- **User-defined variables** User-defined variables are named and defined by the user. They are most often limited in scope and facilitate operations within a specific Storm query.

Built-In Variables

Storm includes a set of built-in variables and associated variable methods and libraries that facilitate Cortex-wide, node-specific, and context-specific operations.

Built-in variables differ from user-defined variables in that built-in variable names:

- are initialized at Cortex start,
- are reserved,
- can be accessed automatically (i.e., without needing to define them) from within Storm, and

- persist across user sessions and Cortex reboots.

Tip: We cover a few of the **most common** built-in variables here. For additional detail on Synapse's Storm types (objects) and libraries, see the *Storm Library Documentation*.

Global Variables

Global variables operate independently of any node. That is, they can be invoked in a Storm query in the absence of any nodes in the Storm execution pipeline (though they can also be used when performing operations on nodes).

$lib

The library variable (`$lib`) is a built-in variable that provides access to the global Storm library. In Storm, libraries are accessed using built-in variable names (e.g., `$lib.print()`).

Libraries provide access to a wide range of additional functionality with Storm. See the *Storm Libraries* technical documentation for descriptions of the libraries available within Storm.

Node-Specific Variables

Storm includes node-specific variables that are designed to operate on or in conjunction with nodes and require one or more nodes in the Storm pipeline.

Note: Node-specific variables are always non-runtsafe.

$node

The node variable (`$node`) is a built-in Storm variable that **references the current node in the Storm pipeline.** Specifically, this variable contains the inbound node's node object, and provides access to the node's attributes, properties, and associated attribute and property values.

Invoking this variable during a Storm query is useful when you want to:

- access the entire raw node object,
- store the value of the current node before pivoting to another node, or
- use an aspect of the current node in subsequent query operations.

The `$node` variable supports a number of built-in **methods** that can be used to access specific data or properties associated with a node. See the technical documentation for the *node* object or the *$node* section of the *Storm Reference - Advanced - Methods* user documentation for additional detail and examples.

$path

The path variable ($path) is a built-in Storm variable that **references the path of a node as it travels through the pipeline of a Storm query.**

The $path variable is not used on its own, but in conjunction with its methods. See the technical documentation for the *node:path* object or the *$path* section of the *Storm Reference - Advanced - Methods* user documentation for additional detail and examples.

Trigger-Specific Variables

A *Trigger* is used to support automation within a Cortex. Triggers use events (such as creating a node, setting a node's property value, or applying a tag to a node) to fire ("trigger") the execution of a predefined Storm query. Storm uses a built-in variable specifically within the context of trigger-initiated Storm queries.

$tag

For triggers that fire on `tag:add` events, the `$tag` variable represents the name of the tag that caused the trigger to fire.

For example:

You write a trigger to fire when any tag matching the expression `#foo.bar.*` is added to a `file:bytes` node. The trigger executes the following Storm command:

```
-> hash:md5 [ +#$tag ]
```

Because the trigger uses a tag glob ("wildcard") expression, it will fire on any tag that matches that expression (e.g., `#foo.bar.hurr`, `#foo.bar.derp`, etc.). The Storm snippet above will take the inbound `file:bytes` node, pivot to the file's associated MD5 node (`hash:md5`), and apply the same tag that fired the trigger to the MD5.

See the *Triggers* section of the *Storm Reference - Automation* document and the Storm *trigger* command for a more detailed discussion of triggers and associated Storm commands.

Ingest Variables

Synapse's *csvtool* can be used to ingest (import) data into Synapse from a comma-separated value (CSV) file. Storm includes a built-in variable to facilitate bulk data ingest using CSV.

$rows

The `$rows` variable refers to the set of rows in a CSV file. When ingesting data into Synapse, CSVTool (or the Optic Ingest Tool) reads a CSV file and a file containing a Storm query that tells Synapse how to process the CSV data. The Storm query is typically constructed to iterate over the set of rows (`$rows`) using a *For Loop* that uses user-defined variables to reference each field (column) in the CSV data.

For example:

```
for ($var1, $var2, $var3, $var4) in $rows { <do stuff> }
```

Tip: The commercial Synapse UI (*Optic*) includes an Ingest Tool that can ingest data in CSV, JSONL, or JSON format. The `$rows` variable is used in the Ingest Tool to refer to either the set of rows in a CSV file or the set of lines ("rows") in a JSONL file. In addition, the `$blob` variable is used to refer to the entire JSON blob when ingesting JSON data. See the ingest examples section of the Ingest Tool documentation for additional detail.

User-Defined Variables

User-defined variables can be defined in one of two ways:

- At runtime (i.e., within the scope of a specific Storm query). This is the most common use for user-defined variables.
- Mapped via options passed to the Storm runtime (for example, when using the *Cortex* API). This method is less common for everyday users. When defined in this manner, user-defined variables will behave as though they are built-in variables that are runtsafe.

Variable Names

All variable names in Storm (including built-in variables) begin with a dollar sign (`$`). A variable name can be any alphanumeric string, **except for** the name of a built-in variable (see *Built-In Variables*), as those names are reserved. Variable names are case-sensitive; the variable `$MyVar` is different from `$myvar`.

Note: Storm will not **prevent** you from using the name of a built-in variable to define a variable (such as `$node = 7`). However, doing so may result in undesired effects or unexpected errors due to the variable name collision.

Defining Variables

Within Storm, a user-defined variable is defined using the syntax:

```
$<varname> = <value>
```

The variable name must be specified first, followed by the equals sign and the value of the variable itself.

`<value>` can be:

- an explicit value (literal),
- a node property (secondary or universal),
- a built-in variable or method (e.g., can allow you to access a node's primary property, form name, or other elements),
- a tag (allows you to access timestamps associated with a tag),
- a library function,
- an expression, or
- an embedded Storm query.

Examples

The examples below use the `$lib.print()` library function to display the **value** of the user-defined variable being set. (This is done for illustrative purposes only; `$lib.print()` is not required in order to use variables or methods.)

In some instances we include a second example to illustrate how a particular kind of variable assignment might be used in a real-world scenario. While we have attempted to use relatively simple examples for clarity, some examples may leverage additional Storm features such as subqueries, subquery filters, or control flow elements such as for loops or switch statements.

Tip: Keep Storm's operation chaining, pipeline, and node consumption aspects in mind when reviewing the following examples. When using `$lib.print()` to display the value of a variable, the queries below will:

- Lift the specified node(s).
- Assign the variable. Note that assigning a variable has no impact on the nodes themselves.
- Print the variable's value using `$lib.print()`.
- Return any nodes still in the pipeline. Because variable assignment doesn't impact the node(s) or transform the working set, the nodes remain in the pipeline and are returned (displayed) at the CLI.

The effect of this process is that for each node in the Storm query pipeline, the output of `$lib.print()` is displayed, followed by the relevant node.

In some examples the Storm *spin* command is used to suppress display of the node itself. We do this for cases where displaying the node detracts from illustrating the value of the variable.

Explicit values / literals

You can assign an explicit, unchanging value to a variable.

- Assign the value 5 to the variable `$threshold`:

```
storm> $threshold=5 $lib.print($threshold)
5
```

Example:

- Tag `file:bytes` nodes that have a number of AV signature hits higher than a given threshold for review:

```
storm> $threshold=5 file:bytes +{ -> it:av:filehit } >= $threshold [ +#review ]
file:bytes=sha256:00007694135237ec8dc5234007043814608f239befdfc8a61b992e4d09e0cf3f
        :sha256 = 00007694135237ec8dc5234007043814608f239befdfc8a61b992e4d09e0cf3f
        .created = 2024/04/19 14:27:01.030
        #review
```

Tip: The example above uses a subquery filter (*Subquery Filters*) to pivot to the `it:av:filehit` nodes associated with the `file:bytes` node, and compares the number of AV hits to the value of the `$threshold` variable.

Node properties

You can assign the value of a particular node property (secondary or universal) to a variable.

- **Secondary property:** Assign the `:user` property from an Internet-based account (`inet:web:acct`) to the variable `$user`:

```
storm> inet:web:acct=(twitter.com,hacks4cats) $user=:user $lib.print($user)
hacks4cats
inet:web:acct=twitter.com/hacks4cats
        :email = ron@protonmail.com
        :site = twitter.com
        :user = hacks4cats
        .created = 2024/04/19 14:27:01.160
```

- **Universal property:** Assign the `.seen` universal property from a DNS A node to the variable `$time`:

```
storm> inet:dns:a=(woot.com,1.2.3.4) $time=.seen $lib.print($time)
(1543289294000, 1565893967000)
inet:dns:a=('woot.com', '1.2.3.4')
        :fqdn = woot.com
        :ipv4 = 1.2.3.4
        .created = 2024/04/19 14:27:01.221
        .seen = ('2018/11/27 03:28:14.000', '2019/08/15 18:32:47.000')
```

Note: In the output above, the variable value is displayed as a pair of epoch milliseconds, which is how Synapse stores date/time values.

Example:

- Given a DNS A record observed within a specific time period, find other DNS A records that pointed to the same IP address in the same time window:

```
storm> inet:dns:a=(woot.com,1.2.3.4) $time=.seen -> inet:ipv4 -> inet:dns:a +.seen@=$time
inet:dns:a=('woot.com', '1.2.3.4')
        :fqdn = woot.com
        :ipv4 = 1.2.3.4
        .created = 2024/04/19 14:27:01.221
        .seen = ('2018/11/27 03:28:14.000', '2019/08/15 18:32:47.000')
inet:dns:a=('hurr.net', '1.2.3.4')
        :fqdn = hurr.net
        :ipv4 = 1.2.3.4
        .created = 2024/04/19 14:27:01.274
        .seen = ('2018/12/09 06:02:53.000', '2019/01/03 11:27:01.000')
```

Tip: An interval (such as a `.seen` property) consists of a **pair** of date/time values. In the example above, the value of the variable `$time` is the combined pair (min / max) of times.

To access the “first seen” (minimum) or “last seen” (maximum) time values separately, use a pair of variables in the assignment:

```
($min, $max) = .seen
```

Built-in variables and methods

Built-In Variables (including *Node-Specific Variables*) allow you to reference common Synapse objects and their associated components. For many common user-facing tasks, the `$node` variable and its methods are the most useful.

- **Node object:** Assign an entire FQDN node to the variable `$fqdn` using the `$node` built-in variable:

```
storm> inet:fqdn=mail.mydomain.com $fqdn=$node $lib.print($fqdn)
Node{(('inet:fqdn', 'mail.mydomain.com'), {'iden':
↪'6511121afd61bf42cb4d14aed4f61daf62ebfc76042dba12d95a6506dd8b6cc4', 'tags': {}, 'props
↪': {'.created': 1713536821328, 'host': 'mail', 'domain': 'mydomain.com', 'issuffix': 0,
↪ 'iszone': 0, 'zone': 'mydomain.com'}, 'tagprops': {}, 'nodedata': {}})}
inet:fqdn=mail.mydomain.com
        :domain = mydomain.com
        :host = mail
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:27:01.328
```

Note: When you use the built-in variable `$node` to assign a value to a variable, the value is set to the **entire node object** (refer to the output above). For common user-facing tasks, it is less likely that users will need "the entire node"; more often, they need to refer to a **component** of the node, such as its primary property value, form name, or associated tags.

For some use cases, Synapse and Storm can "understand" which component of the node you want when referring to the full `$node` object. However, you can always be explicit by using the appropriate **method** to access the component you want (such as `$node.value()` or `$node.form()`).

See the technical documentation for the *node* object or the *$node* section of the *Storm Reference - Advanced - Methods* user documentation for additional detail and examples when using methods associated with the `$node` built-in variable.

- **Node method:** Assign the **primary property value** of a domain node to the variable `$fqdn` using the `$node.value()` method:

```
storm> inet:fqdn=mail.mydomain.com $fqdn=$node.value() $lib.print($fqdn)
mail.mydomain.com
inet:fqdn=mail.mydomain.com
        :domain = mydomain.com
        :host = mail
        :issuffix = false
        :iszone = false
        :zone = mydomain.com
        .created = 2024/04/19 14:27:01.328
```

- Find the DNS A records associated with a given domain where the PTR record for the IP matches the FQDN:

```
storm> inet:fqdn=mail.mydomain.com $fqdn=$node.value() -> inet:dns:a +{ -> inet:ipv4␣
↪+:dns:rev=$fqdn }
inet:dns:a=('mail.mydomain.com', '25.25.25.25')
        :fqdn = mail.mydomain.com
        :ipv4 = 25.25.25.25
        .created = 2024/04/19 14:27:01.381
```

Tip: The example above uses a subquery filter (see *Subquery Filters*) to pivot from the DNS A records to associated IPv4 nodes (`inet:ipv4`) and checks whether the `:dns:rev` property matches the FQDN in the variable `$fqdn`.

Tags

Recall that tags are both **nodes** (`syn:tag=my.tag`) and **labels** that can be applied to other nodes (`#my.tag`). Tags can also have optional timestamps (a time interval) associated with them.

There are various ways to assign tags as variables, depending on what part of the tag you want to access. Many of these use cases are covered above so are briefly illustrated here.

- **Tag value:** Assign an explicit tag value (literal) to the variable `$mytag`:

```
storm> $mytag=cno.infra.dns.sinkhole
```

- **Tag on a node:** Given a `hash:md5` node, assign any malware tags (tags matching the glob pattern `cno.mal.*`) to the variable `$mytags` using the `$node.tags()` method:

```
storm> hash:md5=d41d8cd98f00b204e9800998ecf8427e $mytags=$node.tags(cno.mal.*) $lib.
↪print($mytags)
['cno.mal.foo', 'cno.mal.bar']
hash:md5=d41d8cd98f00b204e9800998ecf8427e
        .created = 2024/04/19 14:27:01.478
        #cno.mal.bar
        #cno.mal.foo
        #cno.threat.baz
```

Tip: In the example above, the value of the variable `$mytags` is the **set** of two tags, `cno.mal.foo` and `cno.mal.bar`, because the MD5 hash node has two tags that match the pattern `cno.mal.*`.

To assign the set of any / all tags on a node to a variable, simply use `$mytags=$node.tags()`.

Note that you can also use `$node.tags()` directly (this method **always** refers to the full set of tags on the current node) without explicitly assigning a separate variable.)

Where the value of a variable is a **set**, a *For Loop* is often used to "do something" based on each value in the set.

Example

- Given an MD5 hash, copy any `cno.mal.*` tags from the hash to the associated file (`file:bytes` node):

```
storm> hash:md5=d41d8cd98f00b204e9800998ecf8427e $mytags=$node.tags(cno.mal.*) for $tag␣
↪in $mytags { -> file:bytes [ +#$tag ] }
file:bytes=sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
        :md5 = d41d8cd98f00b204e9800998ecf8427e
        :sha1 = da39a3ee5e6b4b0d3255bfef95601890afd80709
        :sha256 = e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
        :size = 0
        .created = 2024/04/19 14:27:01.526
        #cno.mal.foo
file:bytes=sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
        :md5 = d41d8cd98f00b204e9800998ecf8427e
        :sha1 = da39a3ee5e6b4b0d3255bfef95601890afd80709
        :sha256 = e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
        :size = 0
        .created = 2024/04/19 14:27:01.526
        #cno.mal.bar
        #cno.mal.foo
```

The output above includes two "copies" of the `file:bytes` node because the node is output twice - once for each iteration of the for loop. The first iteration copies / applies the `cno.mal.foo` tag; the second iteration applies the `cno.mal.bar` tag. For a detailed explanation of this behavior, see *Advanced Storm - Example*.

Tip: The above example explicitly creates and assigns the variable `$mytags` and then uses that variable in a *For Loop*. In this case you can shorten the syntax by skipping the explicit variable assignment and using the `$node.tags()` method directly:

```
hash:md5=d41d8cd98f00b204e9800998ecf8427e for $tag in $node.tags(cno.mal.*) { ->␣
↪file:bytes [ +#$tag ] }
```

- **Tag timestamps:** Assign the times associated with Threat Group 20's control of a malicious domain to the variable `$time`:

```
storm> inet:fqdn=evildomain.com $time=#cno.threat.t20.own $lib.print($time)
(1567900800000, 1631059200000)
inet:fqdn=evildomain.com
        :domain = com
        :host = evildomain
        :issuffix = false
        :iszone = true
        :zone = evildomain.com
        .created = 2024/04/19 14:27:01.575
        #cno.threat.t20.own = (2019/09/08 00:00:00.000, 2021/09/08 00:00:00.000)
```

Example

- Find DNS A records for any subdomain associated with a Threat Group 20 FQDN (zone) during the time they controlled the domain:

```
storm> inet:fqdn#cno.threat.t20.own $time=#cno.threat.t20.own -> inet:fqdn:zone ->␣
↪inet:dns:a +.seen@=$time
inet:dns:a=('www.evildomain.com', '1.2.3.4')
        :fqdn = www.evildomain.com
        :ipv4 = 1.2.3.4
        .created = 2024/04/19 14:27:01.623
        .seen = ('2020/07/12 00:00:00.000', '2020/12/13 00:00:00.000')
inet:dns:a=('smtp.evildomain.com', '5.6.7.8')
        :fqdn = smtp.evildomain.com
        :ipv4 = 5.6.7.8
        .created = 2024/04/19 14:27:01.631
        .seen = ('2020/04/04 00:00:00.000', '2020/08/02 00:00:00.000')
```

Library Functions

Storm types (Storm objects) and Storm libraries allow you to inspect, edit, and otherwise work with data in Synapse in various ways. You can assign a value to a variable based on the output of a method or library.

A full discussion of this topic is outside of the scope of this user guide. See *Storm Library Documentation* for additional details.

- Assign the current time to the variable `$now` using `$lib.time.now()`:

```
storm> $now=$lib.time.now() $lib.print($now)
1713536821685
```

- Convert an epoch milliseconds integer into a human-readable date/time string using `$lib.str.format()`:

```
storm> $now=$lib.time.now() $time=$lib.time.format($now, '%Y/%m/%d %H:%M:%S') $lib.print(
↪$time)
2024/04/19 14:27:01
```

Expressions

You can assign a value to a variable based on the computed value of an expression:

- Use an expression to increment the variable `$x`:

```
storm> $x=5 $x=($x + 1) $lib.print($x)
6
```

Embedded Storm query

You can assign a value to a variable based on the output of a Storm query. To denote the Storm query to be evaluated, enclose the query in curly braces (`{ <storm query> }`).

- Assign an `ou:org` node's guid value to the variable `$org` by lifting the associated org node using its `:name` property:

```
storm> $org={ ou:org:name=vertex } $lib.print($org)
9b848a85ab49f0dc70cce73214e45fa1
```

3.7.2 Storm Reference - Advanced - Methods

Some of Storm's *Built-In Variables* support **methods** used to perform various actions on the object represented by the variable.

A **subset** of the built-in variables / objects that support methods, along with a few commonly used methods and examples, are listed below. For full detail, refer to the *Storm Types* technical reference.

The built-in *$lib* variable is used to access Storm libraries. See the *Storm Libraries* technical reference for additional detail on available libraries.

Note: In the examples below, the `$lib.print()` library function is used to display the value returned when a specific built-in variable or method is called. This is done for illustrative purposes only; `$lib.print()` is not required in order to use variables or methods.

In some examples the Storm *spin* command is used to suppress display of the node itself. We do this for cases where displaying the node detracts from illustrating the value of the variable.

In some instances we have included "use-case" examples, where the variable or method is used in a sample query to illustrate a possible practical use. These represent exemplar Storm queries for how a variable or method might be used in practice. While we have attempted to use relatively simple examples for clarity, some examples may leverage additional Storm features such as subqueries, subquery filters, or control flow elements such as for loops or switch statements.

$node

$node is a built-in Storm variable that references **the current node in the Storm query pipeline.** `$node` can be used as a variable on its own or with the example methods listed below. See the *node* section of the *Storm Types* technical documentation for a full list.

Note: As the `$node` variable and related methods reference the current node in the Storm pipeline, any Storm logic referencing `$node` will fail to execute if the pipeline does not contain a node (i.e., based on previously executing Storm logic).

Examples

- Print the value of `$node` for an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node) | spin
Node{(('inet:dns:a', ('woot.com', 917309932)), {'iden':
↪'01235b5877954084e798f09ba3fd3f1cda2e7b41d79b752b80acbed1b609cbaa', 'tags': {}, 'props
↪': {'.created': 1713536804820, 'fqdn': 'woot.com', 'ipv4': 917309932, '.seen':␣
↪(1482957991000, 1482957991001)}, 'tagprops': {}, 'nodedata': {}})}
```

- Print the value of `$node` for an `inet:fqdn` node with tags present:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node) | spin
Node{(('inet:fqdn', 'aunewsonline.com'), {'iden':
↪'53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4', 'tags': {'rep':␣
↪(None, None), 'rep.mandiant': (None, None), 'rep.mandiant.apt1': (None, None), 'cno':␣
↪(None, None), 'cno.infra': (None, None), 'cno.infra.dns': (None, None), 'cno.infra.dns.
↪sink': (None, None), 'cno.infra.dns.sink.hole': (None, None), 'cno.infra.dns.sink.hole.
↪kleissner': (1385424000000, 1480118400000)}, 'props': {'.created': 1713536804941, 'host
↪': 'aunewsonline', 'domain': 'com', 'issuffix': 0, 'iszone': 1, 'zone': 'aunewsonline.
↪com'}, 'tagprops': {}, 'nodedata': {}})}
```

Note: The value of `$node` is the entire node object and associated properties and tags, as opposed to a specific aspect of the node, such as its iden or primary property value.

As demonstrated below, some node constructors can "intelligently" leverage the relevant aspects of the full node object (the value of the `$node` variable) when creating new nodes.

- Use the `$node` variable to create multiple whois name server records (`inet:whois:recns`) for the name server `ns1.somedomain.com` from a set of inbound whois record nodes for the domain `woot.com`:

```
storm> inet:whois:rec:fqdn=woot.com [ inet:whois:recns=(ns1.somedomain.com,$node) ]
inet:whois:recns=('ns1.somedomain.com', ('woot.com', '2019/06/13 00:00:00.000'))
        :ns = ns1.somedomain.com
        :rec = ('woot.com', '2019/06/13 00:00:00.000')
        :rec:asof = 2019/06/13 00:00:00.000
        :rec:fqdn = woot.com
        .created = 2024/04/19 14:26:45.048
inet:whois:rec=('woot.com', '2019/06/13 00:00:00.000')
        :asof = 2019/06/13 00:00:00.000
        :fqdn = woot.com
        :text = ns1.somedomain.com
```

(continues on next page)

(continued from previous page)

```
        .created = 2024/04/19 14:26:45.009
inet:whois:recns=('ns1.somedomain.com', ('woot.com', '2019/09/12 00:00:00.000'))
        :ns = ns1.somedomain.com
        :rec = ('woot.com', '2019/09/12 00:00:00.000')
        :rec:asof = 2019/09/12 00:00:00.000
        :rec:fqdn = woot.com
        .created = 2024/04/19 14:26:45.051
inet:whois:rec=('woot.com', '2019/09/12 00:00:00.000')
        :asof = 2019/09/12 00:00:00.000
        :fqdn = woot.com
        :text = ns1.somedomain.com
        .created = 2024/04/19 14:26:45.016
```

In the example above, the *$node.value()* method could have been used instead of `$node` to create the `inet:whois:recns` nodes. In this case, the node constructor knows to use the primary property value from the `inet:whois:rec` nodes to create the `inet:whois:recns` nodes.

$node.form()

The `$node.form()` method returns the **form** of the current node in the Storm pipeline.

The method takes no arguments.

Examples

- Print the form of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.form()) | spin
inet:dns:a
```

$node.globtags()

The `$node.globtags()` method returns a **list of string matches from the set of tags applied to the current node** in the Storm pipeline.

The method takes a single argument consisting of a wildcard expression for the substring to match.

- The argument requires at least one wildcard (`*`) representing the substring(s) to match.
- The method performs an **exclusive match** and returns **only** the matched substring(s), not the entire tag containing the substring match.
- The wildcard (`*`) character can be used to match full or partial tag elements.
- Single wildcards are constrained by tag element boundaries (i.e., the dot (`.`) character). Single wildcards can match an entire tag element or a partial string within an element.
- The double wildcard (`**`) can be used to match across any number of tag elements; that is, the double wildcard is not constrained by the dot boundary.
- If the string expression starts with a wildcard, it must be enclosed in quotes in accordance with the use of *Entering Literals*.

See *$node.tags()* to access full tags (vs. tag substrings).

Examples

- Print the set of top-level (root) tags from any tags applied to the current node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.globtags("*")) | spin
['rep', 'cno']
```

- Print the list of numbers associated with any threat group tags (e.g., such as `cno.threat.t42.own` or `cno.threat.t127.use`) applied to the current node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.globtags(cno.threat.t*)) | spin
['83']
```

In the example above, `$node.globtags()` returns the matching substring only ("83"), which is the portion matching the wildcard; it does not return the "t" character.

- Print the list of organizations and associated names (e.g., threat group or malware family names) from any third-party ("rep") tags applied to the current node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.globtags(rep.*.*)) | spin
[('mandiant', 'apt1'), ('symantec', 'commentcrew'), ('mcafee', 'commentcrew'), (
↪'crowdstrike', 'commentpanda')]
```

- Print all sub-tags for any tags starting with "foo" applied to the current node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.globtags(foo.**)) | spin
['bar', 'bar.baz', 'derp']
```

$node.iden()

The `$node.iden()` method returns the *Iden* of the current node in the Storm pipeline.

The method takes no arguments.

Examples

- Print the iden of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.iden()) | spin
01235b5877954084e798f09ba3fd3f1cda2e7b41d79b752b80acbed1b609cbaa
```

$node.isform()

The `$node.isform()` method returns a Boolean value (true / false) for whether the current node in the Storm pipeline is of a specified form.

The method takes a single argument of a form name.

Examples

- Print the Boolean value for whether a node is an `inet:dns:a` form:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.isform(inet:dns:a)) | spin
true
```

- Print the Boolean value for whether a node is an `inet:fqdn` form:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.isform(inet:fqdn))  | spin
false
```

$node.ndef()

The `$node.ndef()` method returns the *Ndef* ("node definition") of the current node in the Storm pipeline.

The method takes no arguments.

Examples

- Print the ndef of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.ndef()) | spin
('inet:dns:a', ('woot.com', 917309932))
```

$node.repr()

The `$node.repr()` method returns the human-friendly *Repr* ("representation") of the specified property of the current node in the Storm pipeline (as opposed to the raw value stored by Synapse).

The method can optionally take one argument.

- If no arguments are provided, the method returns the repr of the node's primary property value.
- If an argument is provided, it should be the string of the secondary property name (i.e., without the leading colon (`:`) from relative property syntax).
- If a universal property string is provided, it must be preceded by the dot / period (`.`) and enclosed in quotes in accordance with the use of *Entering Literals*.

See *$node.value()* to return the raw value of a property.

Examples

- Print the repr of the primary property value of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.repr())  | spin
('woot.com', '54.173.9.236')
```

- Print the repr of the `:ipv4` secondary property value of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.repr(ipv4)) | spin
54.173.9.236
```

- Print the repr of the `.seen` universal property value of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.repr(".seen")) | spin
('2016/12/28 20:46:31.000', '2016/12/28 20:46:31.001')
```

$node.tags()

The `$node.tags()` method returns a **list of the tags applied to the current node** in the Storm pipeline.

The method can optionally take one argument.

- If no arguments are provided, the method returns the full list of all tags applied to the node.
- An optional argument consisting of a wildcard string expression can be used to match a subset of tags.
  - If a string is used with no wildcards, the string must be an exact match for the tag element.
  - The wildcard (`*`) character can be used to match full or partial tag elements.
  - The method performs an **inclusive match** and returns the full tag for all tags that match the provided expression.
  - Single wildcards are constrained by tag element boundaries (i.e., the dot (`.`) character). Single wildcards can match an entire tag element or a partial string within an element.
  - The double wildcard (`**`) can be used to match across any number of tag elements; that is, the double wildcard is not constrained by the dot boundary.
  - If the string expression starts with a wildcard, it must be enclosed in quotes in accordance with the use of *Entering Literals*.

See *$node.globtags()* to access tag substrings (vs. full tags).

Examples

- Print the list of all tags associated with an `inet:fqdn` node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.tags()) | spin
['rep', 'rep.mandiant', 'rep.mandiant.apt1', 'cno', 'cno.infra', 'cno.infra.dns', 'cno.
↪infra.dns.sink', 'cno.infra.dns.sink.hole', 'cno.infra.dns.sink.hole.kleissner', 'rep.
↪symantec', 'rep.symantec.commentcrew', 'rep.mcafee', 'rep.mcafee.commentcrew', 'rep.
↪crowdstrike', 'rep.crowdstrike.commentpanda', 'cno.threat', 'cno.threat.t83', 'cno.
↪threat.t83.own', 'foo', 'foo.bar', 'foo.bar.baz', 'faz', 'faz.baz', 'foo.derp']
```

- Print the tag that exactly matches the string “cno” if present on an `inet:fqdn` node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.tags(cno)) | spin
['cno']
```

- Print the list of all tags two elements in length that start with “foo”:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.tags(foo.*)) | spin
['foo.bar', 'foo.derp']
```

- Print the list of all tags of any length that start with “f”:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.tags(f**)) | spin
['foo', 'foo.bar', 'foo.bar.baz', 'faz', 'faz.baz', 'foo.derp']
```

- Print the list of all tags of any length whose first element is “rep” and whose third element starts with “comment”:

```
storm> inet:fqdn=aunewsonline.com $lib.print($node.tags(rep.*.comment*)) | spin
['rep.symantec.commentcrew', 'rep.mcafee.commentcrew', 'rep.crowdstrike.commentpanda']
```

$node.value()

The `$node.value()` method returns the raw value of the primary property of the current node in the Storm pipeline.

The method takes no arguments.

See *$node.repr()* to return the human-friendly value of a property.

Note: The `$node.value()` method is only used to return the primary property value of a node. Secondary property values can be accessed via a user-defined variable (i.e., `$myvar = :<prop>`).

Examples

- Print the value of the primary property value of an `inet:dns:a` node:

```
storm> inet:dns:a=(woot.com,54.173.9.236) $lib.print($node.value()) | spin
('woot.com', 917309932)
```

$path

$path is a built-in Storm variable that **references the path of a node as it travels through the pipeline of a Storm query.**

The `$path` variable is generally not used on its own, but in conjunction with its methods. See the *node:path* section of the *Storm Types* technical documentation for a full list.

$path.idens()

The `$path.idens()` method returns the list of idens (*Iden*) of each node in a node's path through a Storm query.

The method takes no arguments.

Examples

- Print the list of iden(s) for the path of a single lifted node:

```
storm> inet:fqdn=aunewsonline.com $lib.print($path.idens()) | spin
['53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4']
```

Note: A lift operation contains no pivots (i.e., no "path"), so the method returns only the iden of the lifted node.

- Print the list of idens for the path of a single node through two pivots to a single end node:

```
storm> inet:fqdn=aunewsonline.com -> inet:dns:a +:ipv4=67.215.66.149 -> inet:ipv4 $lib.
↪print($path.idens())
['53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4',
↪'07c79039d00b4391699c9328dc6ccaf864d84d0b38545ded117d1d7ccc6e366c',
↪'9596f5253f25ee74689157706ddf3b459874a6d3cb0adfce4e07018ec8162fc1']
inet:ipv4=67.215.66.149
        :type = unicast
        .created = 2024/04/19 14:26:45.513
```

The example above returns the idens of the original `inet:fqdn` node, the `inet:dns:a` node with the specified IP, and the `inet:ipv4` node.

- Print the list of idens for the path of a single node through two pivots to three different end nodes (i.e., three paths):

```
storm> inet:fqdn=aunewsonline.com -> inet:dns:a -> inet:ipv4 $lib.print($path.idens())
['53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4',
↪'07c79039d00b4391699c9328dc6ccaf864d84d0b38545ded117d1d7ccc6e366c',
↪'9596f5253f25ee74689157706ddf3b459874a6d3cb0adfce4e07018ec8162fc1']
inet:ipv4=67.215.66.149
        :type = unicast
        .created = 2024/04/19 14:26:45.513
['53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4',
↪'0dde48198d3bcc58b40ab82155b218ecd48b533b964d5d2fa3e7453d990541f5',
↪'5af9ae36456988c24edecafa739da75231c067ba3d104a2746e9616ea7a312d6']
inet:ipv4=184.168.221.92
        :type = unicast
        .created = 2024/04/19 14:26:45.520
['53aa7a2f7125392302c36247b97569dd84a7f3fe9e92eb99abd984349dc53fe4',
↪'1c53655a7f3bc67be338cde70d6565d4bc84d343d37513679d4efcd0ec59d3fe',
↪'acecd1f87d1dfc31148bf0ed417b69fde1c77eb2e7effdea434765fe8b759351']
inet:ipv4=104.239.213.7
        :type = unicast
        .created = 2024/04/19 14:26:45.526
```

In the example above, the FQDN has three DNS A records, thus there are three different paths that the original node takes through the query.

3.7.3 Storm Reference - Advanced - Control Flow

Storm includes a number of common programming control flow structures to facilitate more advanced Storm queries. These include:

- *Init Block*
- *Fini Block*
- *Empty Block*
- *If-Else Statement*
- *Switch Statement*
- *For Loop*
- *While Loop*
- *Try…Catch Statement*

The examples below are for illustrative purposes. This guide is **not** meant as a Storm programming tutorial. The intent is to introduce Storm users who may not be familiar with programming concepts (or programmers who are learning to program in Storm) to possible use cases and simple examples for these structures. We've included some *Advanced Storm - Tips* and an *Advanced Storm - Example* to provide some pointers and an illustration of how Storm's "pipeline" behavior and control flow structures may interact.

See the following User Guide and Reference sections for additional information:

- *Storm Reference - Advanced - Variables*

- *Storm Reference - Advanced - Methods*
- *Storm Reference - Advanced - Functions*
- *Storm Libraries*
- *Storm Types*

Storm Developers may also wish to refer to the *Synapse Developer Guide*.

Advanced Storm - Tips

Storm Operating Concepts

It is important to keep the high-level *Storm Operating Concepts* in mind when writing Storm queries or code. These concepts are **critical** to writing more advanced Storm - remembering these fundamentals can save you time and headaches trying to debug a Storm query that is not behaving the way you think it should.

Tip: See the *Advanced Storm - Example* below for an illustration of how these concepts may impact your Storm in unexpected ways.

Storm Debugging Tips

A few helpful tips when writing and debugging advanced Storm:

Be aware of your pipeline. That is, understand what node or nodes are in your **current working set** at any point in your query. A significant part of Storm troubleshooting comes down to figuring out that the current working set is not what you think it is.

Be aware of your variables. Storm supports both runtime-safe ("runtsafe") and non-runtime-safe ("non-runtsafe") variables. Non-runtsafe variables have values that may **change** based on the current node in the Storm pipeline. Another significant part of Storm troubleshooting involves understanding the values of any variables at any given point in your Storm code. (See *Variable Concepts* for additional information.)

Operations may execute multiple times. Because each node passes through each operation in a Storm query individually, operations execute more than once (typically once for each node in the pipeline as it passes through that operation). This includes control flow operations, such as for loops! If you don't account for this behavior with control flow operations in particular, it can result in behavior such as:

- An exponentially increasing working set (if each node passing through an operation generates multiple results, and the results are not deduplicated / uniq'ed appropriately).
- A variable that is set by an operation being consistently changed (re-set) for each node passing through the operation (commonly resulting in "last node wins" with respect to variable assignment).
- A variable that **fails** to be set for a node that does **not** pass through the operation where the variable is assigned (resulting in a `NoSuchVar` error).

Use subqueries…but understand how they work. Unlike most Storm operations and commands, subqueries **do not consume** nodes - by default, what goes into a subquery comes out of a subquery, regardless of what happens inside the subquery itself. This means you can use subqueries with advanced Storm to isolate certain operations and keep the "primary" nodes passing through the Storm pipeline consistent. That said, a node still has to pass **into** a subquery for the Storm inside a subquery to run. If your subquery **fails** to execute, it may be because nothing is going in to it.

Use functions to encapsulate operations. As Storm increases in length or complexity, it can become increasingly challenging to keep track of your pipeline and working set. Functions can be used to simplify your Storm by isolating specific Storm logic within a function.

Start small and add to your Storm incrementally. It's easier to verify that smaller Storm queries execute correctly and then build on that code than to try and write a more advanced query all at once and try to figure out where things aren't working.

As with all debugging, print statements are your friend. Scatter $lib.print() or $lib.pprint() statements generously throughout your Storm during testing. You can print message strings at various points during execution:

```
$lib.print("Hey! This worked!")
```

You can print the value of a variable, to check its value at a given point in your query:

```
inet:ipv4=1.2.3.4
$asn=:asn
$lib.print($asn)
```

You can also print values associated with the node(s) in the current working set, using the various methods associated with the `$node` Storm type. (See *Storm Reference - Advanced - Methods* for a user-focused introduction to methods, or *node* in the detailed Storm Libraries / Storm Types documentation for a more technical discussion.)

```
$lib.print($node.ndef())
```

Control Flow Operations

Tip: The examples below are Storm excerpts used to illustrate specific concepts, but do not represent complete Storm queries / Storm code.

Init Block

An **init block** allows you to execute the specified Storm **once** at the beginning of your Storm query, before nodes enter the Storm pipeline. This allows you to use Storm to perform a set of operations a **single** time only.

See also *Fini Block*.

Syntax:

```
init { <storm> }
```

Example:

You want to use an init block to initialize a set of variables that will be used later in the Storm query. Initializing the variables to default values can:

- Explicitly set a variable value up front.
- Specify default values for variables in the event they are **not** set during subsequent execution (e.g., due to a missing node, property, or tag that the variable depends on).
- Initialize variables that will be modified during execution (e.g., lists, sets, tallies, or other 'count' values you expect to change or increment).

```
init {

    $url=https://www.example.com/my_data/
    $threatname=''
    $fqdns=$lib.set()
    $fqdn_count=0
}
```

Fini Block

A **fini block** allows you to execute the specified Storm **once** after all nodes have passed through the Storm pipeline. This allows you to use Storm to perform a set of operations a **single** time at the end of a Storm query.

See also *Init Block*.

Syntax:

```
fini { <storm> }
```

Example:

You have a Storm query that processes a series of `inet:fqdn` nodes, adding nodes that meet certain criteria to a set (specified with the variable `$fqdns`). After processing the nodes, you want to print a message with the total number of nodes in your set (which you stored in the variable `$fqdn_count`) and return the set of nodes.

```
fini {

    $lib.print(`Total count is {$fqdn_count}`)
    return($fqdns)
}
```

Empty Block

An **empty block** allows you to execute the specified Storm when there are no nodes in the pipeline. If there are nodes in the pipeline, the associated Storm will not be executed.

Syntax:

```
empty { <storm> }
```

Example:

You have a function that conditionally creates a set of nodes and you wish to be warned if no nodes have been made.

```
yield $makeSomeNodes()
empty {
    $lib.print("No nodes created")
}
```

If-Else Statement

An **if-else statement** matches inbound objects against a specified condition. If that condition is met, a set of Storm operations are performed. If the condition is not met, a different set of Storm operations are performed. Storm supports the use of `if` by itself; `if-else`; or `if-elif-else`.

Note that the "Storm operations" performed can include **no** operations / "do nothing" if no Storm is provided (e.g., if the associated curly braces are left empty).

If

Syntax:

```
if <condition> { <storm> }
```

If `<condition>` is met, execute the Storm query in the curly braces. If `<condition>` is not met, do nothing. (Note that this is equivalent to an `if` statement followed by an empty `else` statement.)

Note: If `<condition>` is an expression to be evaluated, it must be enclosed in parentheses `( )`. If the expression includes strings, they must be enclosed in single or double quotes.

```
if ( $str = 'Oh hai!' ) { <storm> }
```

Or:

```
if ( :time > $date ) { <storm> }
```

(Where `:time` represents a property on an inbound node.)

If-Else

Syntax:

```
if <condition> { <storm> }
else { <storm> }
```

If `<condition>` is met, execute the associated Storm; otherwise, execute the alternate Storm.

Similar to the `if` example above with no `else` option (or an empty query for `else`), you can have an empty `if` query:

```
if <condition> { }
else { <storm> }
```

If `<condition>` is met, do nothing; otherwise, execute the alternate Storm query.

If-Elif-Else

Syntax:

```
if <condition> { <storm> }
elif <condition> { <storm> }
else { <storm> }
```

If <condition> is met, execute the associated Storm; otherwise, if (else if) the second <condition> is met, execute the associated Storm; otherwise (else) execute the final Storm query.

You can use multiple `elif` statements before the final `else`. `If-elif-else` is helpful because it allows you to handle multiple conditions differently while avoiding "nested" `if-else` statements.

Example:

You have a subscription to a third-party malware service that allows you to download malware binaries via the service's API. However, the service has a query limit, so you don't want to make any unnecessary API requests that might exhaust your limit.

You can use a simple if-else statement to check whether you already have a copy of the binary in your storage Axon before attempting to download it.

```
<inbound file:bytes node(s)>

if $lib.axon.has(:sha256) { }

else { | malware.download }
```

The Storm query above:

- takes an inbound `file:bytes` node;
- checks for the file in the Axon (*$lib.axon.has(sha256)*) using the `:sha256` value of the inbound file;
- if `$lib.axon.has(:sha256)` returns `true` (i.e., we have the file), do nothing (`{ }`);
- otherwise call the `malware.download` service to attempt to download the file.

Note: In the above example, `malware.download` is used as an example Storm command; it does not exist in the base Synapse code.

Switch Statement

A **switch statement** matches inbound objects against a set of specified constants. Depending on which constant is matched, a set of Storm operations is performed. The switch statement can include an optional **default case** to perform a set of Storm operations in the case where none of the explicitly defined constants are matched.

Syntax:

```
<inbound nodes>

switch <constant> {

  <case1>: { <storm> }
  <case2>: { <storm> }
  <case3>: { <storm> }
```

(continues on next page)

(continued from previous page)

```
  *: { <storm for optional default case> }
}
```

Example:

You want to write a macro (see *Macros*) to automatically enrich a set of indicators (i.e., query third-party data sources for additional data). Instead of writing separate macros for each type of indicator, you want a single macro that can take any type of indicator and send it to the appropriate Storm commands.

A switch statement can send your indicators to the correct services based on the kind of inbound node (e.g., the node's form).

```
<inbound nodes>

switch $node.form() {

    "hash:md5": { | malware.service }

    "hash:sha1": { | malware.service }

    "hash:sha256": { | malware.service }

    "inet:fqdn": { | pdns.service | whois.service }

    "inet:ipv4": { | pdns.service }

    "inet:email": { | whois.service }

    *: { $lib.print("{form} is not supported.", form=$node.form()) }
}
```

The Storm query above:

- takes a set of inbound nodes;
- checks the switch conditions based on the form of the node (see *$node.form()*);
- matches the form name against the list of forms;
- handles each form differently (e.g., hashes are submitted to a malware service, domains are submitted to passive DNS and whois services, etc.)
- if the inbound form does not match any of the specified cases, print (*$lib.print(mesg, **kwargs)*) the specified statement (e.g., `"file:bytes is not supported."`).

The default case above is not strictly necessary - any inbound nodes that fail to match a condition will simply pass through the switch statement with no action taken. It is used above to illustrate the optional use of a default case for any non-matching nodes.

Note: the Storm command names used above are examples only and do not exist in the base Synapse code.

For Loop

A **for loop** will iterate over a set of objects, performing the specified Storm operations on each object in the set.

Syntax:

```
for $<var> in $<vars> {

    <storm>
}
```

Note: The user documentation for the Synapse *csvtool* and the Optic Ingest Tool include additional examples of using for loops.

Example:

You routinely apply tags to files (`file:bytes` nodes) to annotate things such as whether the file is associated with a particular malware family (`cno.mal.redtree`) or threat group (`cno.threat.viciouswombat`). When you apply any of these tags to a file, you want to automatically apply those same tags to the file's associated hashes (e.g., `hash:md5`, etc.)

You can use a for loop to iterate over the relevant tags on the file and apply ("push") the same set of tags to the file's hashes. (**Note:** this code could be executed by a **trigger** (see *Triggers*) that fires when the relevant tag(s) are applied.)

```
<inbound file:bytes node(s)>

{ for $tag in $node.tags(cno.**) {

    { :md5 -> hash:md5 [ +#$tag ] }
    { :sha1 -> hash:sha1 [ +#$tag ] }
    { :sha256 -> hash:sha256 [ +#$tag ] }
    { :sha512 -> hash:sha512 [ +#$tag ] }
}}
```

For each inbound node, the for loop:

- Looks for tags on the node that match the specified pattern (`cno.**`)
- For **each** tag that matches the pattern, execute the Storm code to:
  - Pivot from each of the file's hash properties to the associated hash node.
  - Apply the tag to the node.

Because each "pivot and tag" operation is isolated in a *Subquery*, the original `file:bytes` node remains in our Storm pipeline throughout the set of operations.

Note: A for loop will iterate over "all the things" as defined by the for loop syntax. In the example above, a single inbound node may have multiple tags that match the pattern defined by the for loop. This means that the for loop operations will execute once **per matching tag per node** and yield the inbound node (the `file:bytes` node) to the pipeline for each iteration of the for loop.

In other words, for **each** inbound node:

- the first matching tag causes the for loop to execute;
- the loop operations are performed for that tag (i.e., the tag is applied to the associated hashes);
- the `file:bytes` node is yielded from the for loop;

- if there are additional matching tags to process from the inbound node, **repeat the for loop for each tag.**

Recall that a “single” multi-element tag (such as `cno.mal.redtree`) actually represents three tags (`cno`, `cno.mal`, and `cno.mal.redtree`). If an inbound `file:bytes` node has the tag `#cno.mal.redtree`, the for loop will execute **twice** (for the matching tags `cno.mal` and `cno.mal.redtree`) and yield **two** copies of the `file:bytes` node (one for each match / each iteration of the for loop).

This is by design, and is the way Storm variables (specifically, non-runtime safe variables (*Non-Runtsafe*)) and the Storm execution pipeline (see *Storm Operating Concepts*) are intended to work.

See the *Advanced Storm - Example* below for an illustration of how for loops in particular are impacted by Storm’s pipeline behavior.

While Loop

A **while loop** checks inbound nodes against a specified condition and performs the specified Storm operations for as long as the condition is met.

Syntax:

```
while <condition> {

    <storm>
}
```

While loops are more frequently used for developer tasks, such as consuming from Queues; and are less common for day-to-day user use cases.

Try…Catch Statement

A **try…catch statement** allows you to attempt (try) a Storm operation and handle (catch) any errors if they occur. Because Storm’s default behavior is to halt execution when an error occurs, try…catch statements allow for more graceful error handling within Storm. “Catching” an error allows the remainder of your Storm to continue executing.

Tip: Storm supports some basic error handling (allowing you to “warn and continue” vs “error and halt”) specifically when creating nodes and setting properties or tags through the use of the *“Try” Operator*.

Syntax:

```
try {

    <storm>

} catch <name> as err {

    <storm>
}
```

If the Storm in the try block runs without error, the catch block (or blocks) are ignored. If an error occurs, execution of the try block halts (any remaining Storm in the try block is ignored) and flow passes to the appropriate catch block to handle the error. Multiple catch blocks can be used to handle different kinds of errors.

Because the catch block handles the error, any additional Storm (i.e., after the catch block) will continue to execute.

In the catch block above, `<name>` can be the name of a single error type, a set of error types, or the asterisk (`*`) to represent any error. When using multiple catch blocks, the asterisk can be used in the final block as a default case to catch any error not explicitly handled by a previous catch block.

The catch block can return a status (e.g., `return((1))`) or output a warning message (e.g., using `$lib.warn()` - see *$lib.warn(mesg, **kwargs)*).

Example:

You have an "enrich" macro used to send various kinds of nodes to Storm commands that connect to third-party data sources. There is a particular data source that occasionally returns malformed data, which throws an error and causes the entire macro to halt. You want to isolate the Storm command for that vendor within a try. . . catch block so the macro will continue to run if an error is encountered.

```
try {

    | enrich.badvendor

} catch * as err {

   $lib.warn("BadVendor blew up again!")
}
```

Tip: `$lib.raise()` may also be useful for explicitly raising exceptions (see *$lib.raise(name, mesg, **info)*).

Advanced Storm - Example

The example below is meant to provide a more concrete illustration of some of Storm's pipeline behavior when combined with certain control flow operations - specifically, with for loops. Control flow operations such as if-else or switch statements allow you to perform more advanced Storm operations, but still typically represent a single "path" through the pipeline for any given node - even though the **specific** path for a given node may vary depending on the if-else or switch conditions.

With for loops, however, we may execute the same Storm multiple times, which may have unexpected results if you don't keep Storm's pipeline concept in mind.

For Loop - No Subquery

Consider the following query:

```
inet:fqdn=vertex.link
$list = ('foo','bar','baz')

for $item in $list {

    $lib.print($item)
}

$lib.print('And we're done!')
```

The query:

- lifts a single FQDN node;

- defines a list containing three elements, `foo`, `bar`, and `baz`;
- uses a `for` loop to iterate over the list, printing each element;
- prints `And we're done!`

When executed, the query generates the following output:

```
storm> inet:fqdn=vertex.link
  $list = ('foo', 'bar', 'baz')

  for $item in $list {

    $lib.print($item)
  }

  $lib.print("And we're done!")


foo
And we're done!
inet:fqdn=vertex.link
        :domain = link
        :host = vertex
        :issuffix = false
        :iszone = true
        :zone = vertex.link
        .created = 2024/04/19 14:26:37.375
bar
And we're done!
inet:fqdn=vertex.link
        :domain = link
        :host = vertex
        :issuffix = false
        :iszone = true
        :zone = vertex.link
        .created = 2024/04/19 14:26:37.375
baz
And we're done!
inet:fqdn=vertex.link
        :domain = link
        :host = vertex
        :issuffix = false
        :iszone = true
        :zone = vertex.link
        .created = 2024/04/19 14:26:37.375
```

What's going on here? Why does `And we're done!` print three times? Why do we apparently have three copies of our FQDN node? The reason has to do with Storm's pipeline behavior, and how our FQDN node travels through the pipeline when the pipeline loops.

Our query starts with a single `inet:fqdn` node in our initial working set. Setting the `$list` variable does not change our working set of nodes.

When we reach the for loop, the loop needs to execute multiple times (three times in this case, once for each item in `$list`). Anything currently in our pipeline (any nodes that are inbound to the for loop, as well as any variables that are currently set) is passed into **each** iteration of the for loop.

In this case, because the for loop is part of our **main** Storm pipeline (it is not isolated in any way, such as by being placed inside a subquery), **each iteration** of the loop outputs our original FQDN node… which then continues its passage through the remainder of the Storm pipeline, causing the `$lib.print('And we're done!')` statement to print (remember, each node travels through the pipeline one by one). Storm then executes the second iteration of the for loop, and the FQDN that exits from this second iteration continues through the pipeline, and so on.

It may help to think of this process as the for loop effectively "splitting" the main Storm pipeline into multiple pipelines that then each continue to execute in full, one after the other.

Note: Each pipeline still executes **sequentially** - not in parallel. So the first iteration of the for loop (where `$item=foo`) will execute and the remainder of the Storm pipeline will run to completion; followed by the second iteration of the for loop and the remainder of the Storm pipeline, and so on. (This is why one instance of `And we're done!` prints before the messages associated with the second iteration of the loop where `$item=bar`, etc.).

For Loop - With Subquery

In this variation on our original query, we isolate the for loop within a subquery (*Storm Reference - Subqueries*):

```
inet:fqdn=vertex.link
$list = ('foo','bar','baz')

{
    for $item in $list {

        $lib.print($item)
    }
}

$lib.print('And we're done!')
```

The query performs the same actions as described above, but thanks to the subquery, the behavior of this query is different, as we can see from the query's output:

```
storm> inet:fqdn=vertex.link
  $list = ('foo', 'bar', 'baz')

  {
      for $item in $list {
          $lib.print($item)
      }
  }

  $lib.print("And we're done!")


foo
bar
baz
And we're done!
inet:fqdn=vertex.link
        :domain = link
        :host = vertex
```

(continues on next page)

(continued from previous page)

```
        :issuffix = false
        :iszone = true
        :zone = vertex.link
        .created = 2024/04/19 14:26:37.375
```

In this case, the query behaves more "as expected" - the strings within the for loop print once for each item / iteration of the loop, `And we're done!` prints once, and a single FQDN node exits our pipeline when our query completes. So what's different?

One of the key features of a subquery is that by default (i.e., unless the `yield` option is used), **the nodes that go into a subquery also come out of a subquery,** regardless of what occurs inside the subquery itself. In other words, **subqueries do not "consume" nodes**.

We still have our single FQDN inbound to the subquery. Inside the subquery, our for loop still executes, effectively "splitting" the Storm pipeline into three pipelines that execute in sequence. But once we complete the for loop and exit the subquery, those pipelines are "discarded". The single FQDN that went into the subquery exits the subquery. We are back to our single node in the main pipeline. That single node causes our print statement to print `And we're done!` only once, and we are left with our single node at the end of the query.

3.7.4 Storm Reference - Advanced - Functions

This section provides an overview of the types of functions available in Storm, along with some tips, caveats, and basic examples. It is meant to introduce the concepts around functions in Storm; it is **not** meant as a Storm programming tutorial.

Storm Developers can refer to the *Synapse Developer Guide* for additional information.

Overview

Functions can be used to encapsulate a set of Storm logic that executes when the function is invoked. Functions are declared and then invoked within a Storm query using the function name and any required parameters. Separating the function's logic from the logic of your "executing" Storm query makes your Storm cleaner and easier to read (and also allows for easier code reuse).

Functions and the Storm Pipeline

We regularly emphasize the *Storm Operating Concepts*, especially when writing more complex Storm queries and Storm logic. In particular, it is important to pay attention to Storm's **pipeline** behavior and the way the pipeline affects your **working set.**

A function in Storm has its **own node pipeline,** independent of any Storm logic that invokes the function. Functions **do not** inherit the pipeline of the invoking query, and **do not** modify the invoking pipeline by default.

Because the function **itself** is a Storm pipeline, all caveats about pipelines and awareness of your working set still apply to the Storm **within** the function.

All of Storm's features and capabilities are available for use within a function. This includes Storm operations, commands, variables, methods, control flow, libraries, etc.

You can use an init block or a fini block within a function to execute a subset of Storm logic a single time before operating on any nodes in the functions's pipeline or after any nodes have exited the function's pipeline, respectively.

There are two subtle but important aspects of function behavior to keep in mind:

- Nodes are not "sent inbound" to a function to "cause" it to execute; a function runs when it is **invoked** as part of a Storm query (i.e., by an invoking Storm pipeline). This means that:
  - an invoked function can execute even if there are no nodes in the invoking pipeline, as long as the associated Storm logic executes; and
  - a function within an invoking pipeline will run once each time that pipeline executes; and an invoking pipeline may execute multiple times (for example, if multiple nodes are passing through the invoking pipeline). In this case the nodes themselves don't "cause" the function to execute; the pipeline runs once per node, and the function invoked by the pipeline runs once each time the pipeline runs.
- Nodes do not "pass into" functions by default, as the function and the invoking Storm logic are two separate pipelines. It is possible to **invoke** a function so that it operates on a node or nodes; but the function will not "automatically" do so.

The sections below on *Operating on Nodes* and *Runtsafe vs. Non-Runtsafe Functions* discuss these behaviors in more detail.

Function Basics

Storm supports three types of functions. Each is explained in more detail below.

- **callable** functions, which are "regular" functions (similar to functions in other programming languages). Callable functions return a value.
- **data emitter** functions, which emit data.
- **node yielder** functions, which yield nodes.

Declaring Functions

All functions are **declared** in the same way, using the `function` keyword.

- `function myFunction() { <do stuff> }`
- `function myFunction(foo) { <do stuff> }`
- `function myFunction(bar, baz=$lib.null) { <do stuff> }`

Invoking Functions

All functions are **invoked** using the function name preceded by the dollar sign (`$`), and by passing any required parameters to the function.

- `$myFunction()`
- `$myFunction(foo)`
- `$myFunction($foo)`
- `$myFunction(bar, baz=wheee)`

Parameters can be passed as literals, variables, or keyword=$valu arguments. For example, given a function that takes an organization name as input:

```
function $myFunction(orgname) { <do stuff> }
```

The name can be passed directly:

```
$myFunction("The Vertex Project")
```

…as a variable:

```
$name="The Vertex Project"
$myFunction($name)
```

…or as a keyword/value pair:

```
$myFunction(orgname="The Vertex Project")
```

```
$name="The Vertex Project"
$myFunction(orgname=$name)
```

Operating on Nodes

Functions do not inherit or operate on the invoking Storm pipeline by default. If you want a function to operate on nodes in a pipeline, you must invoke the function in such a way as to pass the node (or a property or properties of the node) as input to the function.

For example, if your invoking pipeline consists of a set of `inet:ipv4` nodes, a function can take the `:asn` property as input:

```
$myFunction(:asn)
```

Or:

```
$asn=:asn
$myFunction($asn)
```

Alternatively, you can pass the entire `$node` object to the function and use the `yield` keyword within the function to yield the node into the function's pipeline:

```
//Declare function
function $myFunction(inboundNode) {

    yield $inboundNode
    <do stuff>
    return()
}

//Invoke function
$myFunction($node)
```

If another function yields the node(s) you want to operate on, that function can be used as input to a second function. A simple example:

Function 1 (node yielder):

```
function getIPs() {
    //Lift 10 IPv4 addresses
    inet:ipv4 | limit 10
}
```

Function 2 (callable function):

```
//Takes a generator object as input
function counter(genr) {

    //Yield the generator content into the function pipeline
    yield $genr

    //Print the human-readable representation of each node
    $lib.print($node.repr())

    //Return the output of the function
    fini { return() }
}
```

Function 2 invoked with Function 1 as input:

```
$counter($getIPs())
```

When executed, the function produces the following output. Note that the `$counter()` function simply prints the nodes' human-readable representation (`$node.repr()`) as an example; it does not return or yield the `inet:ipv4` nodes:

```
storm>
function getIPs() {
    inet:ipv4 | limit 10
}

function counter(genr) {
    yield $genr
    $lib.print($node.repr())
    fini { return() }
}

$counter($getIPs())
1.1.1.1
2.2.2.2
3.3.3.3
4.4.4.4
5.5.5.5
6.6.6.6
7.7.7.7
8.8.8.8
9.9.9.9
10.10.10.10
```

Runtsafe vs. Non-Runtsafe Functions

Just as variables may be runtime-safe (runtsafe) or non-runtime-safe (non-runtsafe), functions can be invoked in a runtsafe manner (or not) based on the parameters passed to the function.

If a function is invoked with a runtsafe (typically static) value, the function is considered runtsafe. A function that takes an Autonomous System (AS) number as input and is passed a static AS number as a parameter is invoked in a **runtsafe** manner:

```
$myFunction(9009)
```

Or:

```
$asn=9009
myFunction($asn)
```

If the same function is invoked with a per-node, non-runtsafe value or values, the function is considered **non-runtsafe,** such as the example above where the invoking pipeline contains `inet:ipv4` nodes and the function is invoked with the value of each node's `:asn` property:

```
$myFunction(:asn)
```

Or:

```
$asn=:asn
$myFunction($asn)
```

Tip: Keep in mind that functions execute when they are invoked. This has some implications with respect to runtime safety ("runtsafety"):

- A non-runtsafe function (i.e., that is dependent on a per-node value) will **not** execute when invoked if there are no nodes in the invoking pipeline. Synapse will not generate an error but the function will not "do anything".
- A runtsafe function (i.e., one whose parameters are **not** node-dependent) will still execute once each time it is invoked. If the invoking Storm executes multiple times, this can result in the runtsafe function running repeatedly while simply "doing the same thing" each time (based on its runtsafe input parameters). If the function should only execute once, it can be placed in a fini block (or an init block as appropriate).

Function Output

Functions do not modify the invoking Storm pipeline by default. To access the output of a function (whether nodes, data, or a value), you can:

- Assign the output of the function to a variable:

  ```
  $x = $myFunction()
  ```

- Iterate over the function's output (used with data emitters and node yielders):

  ```
  for $x in $myFunction() { <do stuff> }
  ```

- Add the node or nodes generated by the function directly to the invoking Storm pipeline with the `yield` keyword (used with node yielders and callable functions that return a node):

```
yield $myFunction()
```

Types of Functions

Note: Because all functions in Storm are declared and invoked the same way, the Storm syntax parser relies on the presence (or absence) of specific **keywords** within a function to identify the type of function and how to execute it.

- Callable functions **must** include a `return()` statement (and must not use `emit`).
- Data emitter functions **must** use the `emit` keyword (and must not use `return()`).
- Node yielder functions **must not** include the keywords `emit` or `return()`.

Both data emitters and node yielders may optionally include the keyword `stop` to cleanly halt execution and exit the function. (Using `stop` in a callable function will generate a `StormStop` error.)

Functions can be declared and invoked on their own, but are most often used when authoring more extensive Storm code to implement a set of related functionality, such as a Rapid Power-Up. A set of functions, each encapsulating Storm logic to perform a specific task, can work together to implement more complex capabilities. Given this architecture, it is common for functions to invoke other functions as part of their code, or to take the output of another function as an input parameter to perform another operation, as seen in some of the examples below.

See the *Rapid Power-Up Development* section of the *Synapse Developer Guide* for a more in-depth discussion of how to integrate multiple Storm components into a larger package.

Callable Functions

Callable functions are "regular" functions, similar to those in other programming languages. A callable function is invoked (called) and returns a value using a `return()` statement. A `return()` statement **must** be present for a callable function to execute properly even if the function does not return a specific value.

Callable functions are **executed in their entirety** before returning. They return **exactly** one value.

Tip: Callable functions may contain multiple `return()` statements, based on the function's logic. The **first** `return()` encountered during the function's execution will cause the function to stop execution and return. If you are performing multiple actions within the function and want to ensure they all complete before the function returns, place the `return()` in a fini block so it executes once at the end of the function's pipeline.

Use Cases

Callable functions can be used to:

- Check a condition and return a status (e.g., `return((0))` vs. `return((1))`).
- Return a value (such as a count).
- Return a single node.
- Perform isolated operations on a node in the pipeline.
- Retrieve data from an external API.

Pseudocode

```
function callable() {

    <do stuff>
    return()
}
```

Examples

Return a node

A callable function can take input and attempt to create (or lift) a node.

```
//Takes a value expected to be an IPv4 or IPv6 as input
function makeIP(ip) {

    //Attempt to create (or lift) an IPv4 from the input
    //Return the IPv4 and exit if successful
    [ inet:ipv4 ?= $ip ]
    return($node)

    //Otherwise, atempt to create (or lift) an IPv6 from the input
    //Return the IPv6 and exit if successful
    [ inet:ipv6 ?= $ip ]
    return($node)

    //If the input is not a valid IPv4 or IPv6, the function
    // will execute but will not return a node.
}

//Invoke the function with the specified input and
// yield the result (if any) into the pipeline
yield $makeIP(8.8.8.8)
```

Return a node using secondary property deconfliction

When ingesting or creating guid-based nodes, a common deconfliction strategy is to check for existing nodes using one or more secondary properties (known as secondary property deconfliction). A callable function that takes a secondary property value (or values) as input and returns (or creates) the node simplifies this process.

```
//Create an ou:org node based on an org name (ou:name)

//Declare function - takes 'name' as input
function genOrgByName(name) {

    //Check whether input is valid for an ou:name value
    //If not, return / exit
    ($ok, $name) = $lib.trycast(ou:name, $name)
    if (not $ok) { return() }

    //If name is valid, attempt to identify an existing ou:org
```

(continues on next page)

(continued from previous page)

```
    //Lift the ou:name node for 'name' (if it exists)
    // and pivot to an org with that name (if it exists)
    //Return the existing node if found
    ou:name=$name -> ou:org
    return($node)

    //If an org is not found, create a new ou:org using 'gen' and the name
    // as input for the org's guid; set the :name prop
    //Return the new node
    [ ou:org=(gen, $name) :name=$name ]
    return($node)
}

//Invoke the function with input name "The Vertex Project" and yield
// the result into the pipeline
yield $genOrgByName("The Vertex Project")
```

Tip: Synapse includes gen.* (generator) Storm commands and $lib.gen APIs that can generate many common guid-based forms using secondary property deconfliction.

Return a value

Some data sources provide feed-like APIs that allow you to retrieve either the entire feed or just retrieve any new items added since your last update. The "last update" time can be stored as *Node Data* on the `meta:source` node for the data source. A callable function can retrieve the "last updated" date (e.g., to pass the value to another function used to retrieve only the latest feed data).

```
function getLastReportDate() {

    //Invoke an existing function to create (initialize) or retrieve the meta:source node
    // and yield the node into the function's pipeline
    yield $initMetaSource()

    //Set the $date variable to the value of the node data key mysource:report:date from
    // the meta:source node.
    $date = $node.data.get(mysource:report:date)

    //If there is no value for this key return the integer 0
    if ($date = $lib.null) { return((0)) }

    //Otherwise return the date
    return($date)
}

//Assign the value returned by this function to the variable $date for use by the
// invoking Storm pipeline. This value can be passed to another function that retrieves
// the latest feed data.
$date = $getLastReportDate()
```

Data Emitter Functions

Data emitter functions emit data using the `emit` keyword. The `stop` keyword can optionally be used to halt processing and exit the function. The `emit` keyword **must** be present for a data emitter function to execute properly.

Data emitter functions **stream** data (technically, they return a **generator object** that is iterated over). They are designed to emit data to the invoking pipeline as it is available; they may be invoked with **for** or **while** loops for this purpose. When data is emitted, execution of the function is paused until the invoking pipeline requests the next value, at which point the function's execution resumes.

Use Cases

Data emitter functions can be used to:

- Consume data from sources that paginate results, where you want to mask the pagination (i.e., a data emitter can consume and emit the first page of results; then consume and emit the next page; and so on).
- Consume data from sources that stream results, where the data emitter is used to continue the streaming behavior.

Tip: Data emitters can be used to emit nodes (e.g., `emit $node`), though this is an uncommon use case. The ability of data emitters to emit data incrementally is useful when consuming large result sets from an API. "Subsets" of results (such as individual JSON objects from a JSON blob) can be made available more quickly (e.g., to another function responsible for creating nodes from the JSON) while the emitter continues to process data.

In contrast, if the same set of API results was consumed by a callable function, the function would need to consume the entire result set before returning.

Pseudocode

```
function data_emitter() {

    for $thing in $things {
        <do stuff>
        emit $thing
    }
}
```

Or:

```
function data_emitter() {

    for $thing in $things {
        <do stuff>
        emit $thing

        if ($thing = "badthing") {
            stop
        }
    }
}
```

Or:

```
function data_emitter() {

    while (1) {
        <do stuff>
        emit $thing

        if (<end condition>) { stop }

        <update something to continue while loop>
    }
}
```

Example

Some data sources may paginate results, returning X number of objects (e.g., in a JSON blob) at a time until all results are returned. A data emitter function can emit individual JSON objects from the blob (e.g., for consumption by another function that processes the object and creates nodes) until all of the results have been received.

```
function emitReportFeed() {

    //Set variables for the current time and the # of objects to retrieve per page
    $now = $lib.time.now()
    $pagelim = 100

    //Set a variable for API query parameters
    $params = ({
        "limit": $pagelim,
    })

    //Set a variable for the initial offset
    $offset = (0)

    //While loop to retrieve records
    while (1) {

        //Set the value of the 'offset' parameter
        $params.offset = $offset

        //Invoke an existing function to retrieve the JSON using $params as parameters
        // to the API request.
        //Assign the returned JSON to the variable $data
        $data = $getJson("/reports", params=$params)

        //If no data is returned, stop and exit this function
        if ($data = $lib.null) { stop }

        //If data is returned, loop over the JSON and emit each item / ojbect
        for $item in $data.data { emit $item }

        //Set $datasize to the size (number of items) in the returned JSON
        $datasize = $data.data.size()
```

(continues on next page)

(continued from previous page)

```
        //Check whether the # of records returned is less than our page limit
        //If so we have retrieved all available records
        if ($datasize < $pagelim) {

            //Print status to CLI if debug is in use
            if $lib.debug { $lib.print(`Reports ingested up to {$now}`) }

            //Invoke an existing function to update the 'last retrieved' date to the␣
↪current time
            //E.g., this value may be stored as node data on the feed's meta:source node
            $setReportFeedLast($now)

            //Stop and exit the function
            stop
        }

        //If $datasize is NOT < $pagelim there is more data
        //Update the $offset value and execute the while loop again
        $offset = ($offset + $pagelim)
    }
}
```

Node Yielder Functions

Node yielder functions yield nodes. If a function **does not** include either of the keywords `return` or `emit`, it is presumed to be a node yielder.

Node yielder functions **stream** nodes; (technically, they return a **generator object** that is iterated over). They are designed to yield nodes as they are available while continuing to execute. They may be invoked with the `yield` keyword or with a **for** loop for this purpose.

Use Cases

Node yielder functions can be used to:

- Isolate different node construction pipelines during complex data ingest logic.

Pseudocode

```
function node_yielder() {
    <do stuff>
}
```

Examples

Some data sources allow you to retrieve specific records or reports (e.g., based on a record or report number). A node yielder function can request the record(s) and yield the node(s) created from those records (e.g., a report retrieved from a data source may be used to create a `media:news` node).

```
//Function takes one or more IDs as input
function reportByID(reportids) {

    //Loop over report IDs
    for $reportid in $reportids {

        //Invoke an existing privileged function to retrieve the report object (i.e., a␣
↪JSON response)
        //A privileged module may be invoked to mask sensitive data such as an API key␣
↪from a normal user
        $report = $privsep.getReportById($reportid)

        //Print the JSON to CLI if debug is in use
        if $lib.debug { $lib.pprint($report) }

        //Yield the node (e.g., media:news node) created by invoking an existing␣
↪function that
        // creates the media:news node from the $report
        yield $ingest.addReport($report)
    }
}
```

Functions and Privilege Separation

Functions can be used to support **privilege separation** ("privsep") for things like custom Power-Up development. Storm logic that requires access to sensitive information (such as API keys or other credentials) can be encapsulated in a function that is not accessible to unprivileged users. The function can return non-sensitive data that is "safe" for viewing or consumption.

See the *Rapid Power-Up Development* Guide and in particular the section on privileged modules for more information.

Function Debugging Tips

Functions execute Storm, so standard Storm debugging tips still apply to all code **within** the function itself (and to the Storm code that invokes the function, of course). The following additional tips apply to functions in particular.

Use the right type of function for your use case. Each Storm function serves a different purpose; be clear on what type of function you need for a given situation.

For example, a node yielder can yield multiple nodes. A callable function can also yield multiple nodes (e.g., by returning a set or list object). But there can be significant (even damaging) performance differences between the two, depending on the nature of the function.

A node yielder yields a generator object that can incrementally provide results (i.e., for a streaming effect). When written as a node yielder, a function to lift every node in a Cortex is workable, even for large result sets:

```
function allnodes() { .created }
```

You could write the same function as a callable function, but it would likely blow up your system by consuming all available memory. A callable function can only return exactly one object; it can't stream results. You could write a callable function to lift each node, add it to a set object, and have the function return the set. But the callable function will need to construct and store the **entire set** in memory until the object can be returned:

```
// NEVER DO THIS
function allnodes() {

    $set = ([])
    .created
    $set.add($node)
    fini { return($set) }
}
```

While this is an extreme example, it serves to illustrate some of the differences between function types.

Ensure necessary keywords are present for your function type. Synapse determines "what kind" of function is present and how to execute it based on keywords (e.g., `return()` for callable functions, `emit` for data emitters). If you write a node yielder function with a `return()` statement, Synapse will attempt to execute it as a callable function. Similarly, a callable function that is missing a `return()` will not execute properly.

Note: Data emitters and node yielders may **fail** to emit data or yield nodes, based on the input to the function and the function's code. In these situations it can be challenging to determine whether a function that is "not doing anything" is a yielder / emitter that is failing to produce output, or a callable function that is missing a `return()` statement.

Understand pipeline interactions between functions and Storm logic that invokes them. By default, functions do not interact with the Storm pipeline that invokes them.

If you want a function to operate on nodes in the invoking Storm pipeline, you must invoke the function in such a way as to do this.

Note: If a function is written to operate on or iterate over nodes, and there are no nodes in the pipeline (based on previously executing Storm logic), the function will not execute.

If you want the invoking Storm pipeline to operate on the function's output, you must ensure that the output is returned to the pipeline (e.g., assign the function's output to a variable; use the `yield` keyword to yield any nodes into the pipeline; use a for loop to iterate over function results; etc.).

Many of the concepts above are closely related and this outline represents a reasonable effort to introduce concepts in a logical order. However, it is difficult to fully understand the potential of Synapse without grasping the power of the Storm query language to understand, manipulate, and annotate data. Similarly, it's hard to understand the effectiveness of Storm without knowledge of the underlying data model. The outline above is our suggested order but readers are encouraged to skip around or revisit earlier sections after digesting later sections to better see how these topics are tied together.

CHAPTER

FOUR

SYNAPSE ADMIN GUIDE

This guide is designed for use by Synapse Administrators ("global admins"). Synapse Admins are typically Synapse power-users with `admin=true` privileges on the *Cortex* who are responsible for configuration and management of a production instance of Synapse.

The Synapse Admin Guide provides important instructions and background information on topics related to day-to-day Synapse administrative tasks, and focuses on using *Storm* to carry out those tasks.

Synapse provides a number of additional methods that can be used to perform some or all of the tasks described in this guide; however, these methods are **not** covered here. Additional methods include:

- *Storm Libraries* that allow you to work with a broad range of objects in Synapse.
- Synapse tools that can be used from the host CLI (as opposed to the Storm CLI). Tools are available in the synapse.tools package of the *Synapse Python API*. The *Synapse User Guide* includes documentation on some of these *Tools*.
- The *Synapse HTTP/REST API*.

Tip: If you are a commercial Synapse user with the Synapse UI (Optic), see the UI documentation for information on performing Synapse Admin tasks using Optic. Optic simplifies many of Synapse's administrative tasks. However, we encourage you to review the information in this guide for important background and an overview of the relevant topics.

4.1 Enable Synapse Power-Ups

The Vertex Project provides a number of Power-Ups that extend the functionality of Synapse. For more information on configuring your Cortex to use **Rapid Power-Ups,** see the blog post on Synapse Power-Ups.

Note: **Advanced Power-Ups** are deployed via their own Docker containers and are typically configured by a DevOps team.

4.2 Create and Manage Users and Roles

A *User* account is required to authenticate to and access Synapse. Having "a Synapse account" effectively means having an account in the Cortex.

In Synapse, a *Role* can be used to "group" users with similar responsibilities (and related permissions requirements). You can **grant** or **revoke** one or more roles from a user.

You grant (or deny) **permissions** to users or roles by assigning **rules** that specify those permissions (see *Assign and Manage Permissions*).

Synapse includes the following built-in users and roles:

- **Root** user. The **root** account has *Admin* privileges in the Cortex. The **admin** status of the root account cannot be revoked, and the account cannot be locked / disabled.
- **All** role. The **all** role has **read** access to the Cortex (specifically, to any view with `worldreadable=true`, which includes the **default** view). All user accounts are automatically granted the **all** role (are part of the **all** "group"); this role cannot be revoked.

Tip: The set of Storm *auth* commands are collectively used to manage users, roles, and permissions from Storm.

In the commercial Optic UI, users, roles, and permissions can be managed through the **Admin Tool** and through dialogs associated with various objects (such as Views or Stories).

Note: The descriptions and examples below assume that you have deployed Synapse using native Synapse management and authentication of users, roles, and permissions.

The *Synapse Devops Guide* includes information on provisioning **initial** users when Synapse is first deployed (see *Managing Users and Roles*). This guide focuses on ongoing management of users and roles once Synapse admins have access to Storm (i.e., the Storm CLI or Optic UI).

4.2.1 Working with Users

Add a User

The *auth.user.add* command creates a new user. Newly created users do not have any permissions (other than those associated with the built-in **all** role).

Example:

Add the user "Ron" with email address `ronthecat@vertex.link`:

```
storm> auth.user.add ron --email ronthecat@vertex.link
User (ron) added with iden: 7d7e0dcdf4f5cc44c1c135f954dc7c4f
```

Tip: Users are represented by a unique 128-bit identifier (iden). You can modify information about the user account (such as the username or associated email address) without affecting the underlying identifier or any associated roles or permissions.

Display a User

The *auth.user.show* command displays information about a user, including any assigned roles or rules (permissions) and their order.

Example:

Display information for user “Ron”:

```
storm> auth.user.show ron
User: ron (7d7e0dcdf4f5cc44c1c135f954dc7c4f)

  Locked: false
  Admin: false
  Email: ronthecat@vertex.link
  Rules:

  Roles:
    b07861cadfc09ee2d29bba1b4240e333 - all

  Gates:
```

Modify a User

The *auth.user.mod* command modifies a user account. Use the command to:

- Change the username or email address associated with the user.
- Set or reset the user’s password.
- Assign (or remove) **admin** status for the user.
- Lock (or unlock) the account.

Examples:

Update the email address for user “Ron”:

```
storm> auth.user.mod ron --email ron@vertex.link
User (ron) email address set to ron@vertex.link.
```

Assign **admin** status to the user “ron_admin”:

```
storm> auth.user.mod ron_admin --admin $lib.true
User (ron_admin) admin status set to true.
```

Remove **admin** status from user “ron_admin”:

```
storm> auth.user.mod ron_admin --admin $lib.false
User (ron_admin) admin status set to false.
```

Lock the user account “ron_admin”:

```
storm> auth.user.mod ron_admin --locked $lib.true
User (ron_admin) locked status set to true.
```

Warning: We strongly encourage you to **lock** (disable) accounts when necessary instead of deleting them. Changes to data in the Cortex (such as creating nodes, setting properties, or adding tags) are associated with the user account that made those changes. Deleting an account associated with past changes will prohibit you from identifying the user who made those changes.

If necesssary, user accounts can be deleted using the *$lib.auth.users.del(iden)* library, but there is no equivalent Storm command.

List All Users

The *auth.user.list* command lists all users in the Cortex.

Example:

List all users:

```
storm> auth.user.list
Users:
  ron
  root

Locked Users:
  ron_admin
```

4.2.2 Working with Roles

Add a Role

The *auth.role.add* command creates a new role. Newly created roles do not have any permissions or associated user accounts.

Example:

Add the new role "cattribution analyst":

```
storm> auth.role.add "cattribution analyst"
Role (cattribution analyst) added with iden: 9835c2d9f4240229b0f909b4aadf789d
```

Tip: Roles are represented by a unique 128-bit identifier (iden). You can later change information about the role (such as the role name) without affecting the underlying role or any associated permissions or users.

Display a Role

The *auth.role.show* command displays information about a role, including any assigned rules (permissions) and their associated objects.

Example:

Display information for the "all" role:

```
storm> auth.role.show all
Role: all (b07861cadfc09ee2d29bba1b4240e333)

  Rules:

  Gates:
    1b0a73c2fda8f48bc24d5962a9351821 - (layer)
      [0  ] - layer.read
    e97aedae71082cd66a877fe3a3f08c02 - (view)
      [0  ] - view.read
```

Modify a Role

The *auth.role.mod* command modifies a role. The command can be used to change the name of the role.

Example:

Change the name of the role "cattribution analyst" to "meow-ware analyst":

```
storm> auth.role.mod "cattribution analyst" --name "meow-ware analyst"
Role (cattribution analyst) renamed to meow-ware analyst.
```

List all Roles

The *auth.role.list* command lists all roles in the Cortex.

Example:

List all roles:

```
storm> auth.role.list
Roles:
  a-cat-emic researcher
  all
  cattribution analyst
  meow-ware analyst
```

Delete a Role

The *auth.role.del* command deletes a role.

Example:

Delete the role "meow-ware analyst":

```
storm> auth.role.del "meow-ware analyst"
Role (meow-ware analyst) deleted.
```

Note: Deleting a role has no impact on any users who have been granted the role (other than losing any permissions provided by that role). The user accounts remain intact and the role is simply removed from each user's list of roles.

4.2.3 Grant or Revoke Roles

Granting a role to a user allows the user to inherit the role's permissions. **Revoking** a role removes the associated permissions from the user. It is not possible to grant a role to another role (i.e., roles cannot be nested).

Roles can be granted or revoked using the *auth.user.grant* and *auth.user.revoke* commands.

Examples:

Grant the role "cattribution analyst" to the user "ron":

```
storm> auth.user.grant ron "cattribution analyst"
Granting role cattribution analyst to user ron.
```

Revoke the role "a-cat-emic researcher" from user "ron":

```
storm> auth.user.revoke ron "a-cat-emic researcher"
Revoking role a-cat-emic researcher from user ron.
```

Note: The order in which roles are granted to a user matters; when determining whether a user has permission to perform an action, the permissions for each of the user's roles are checked in sequence.

Each role granted to a user is added to the **end** of the set of roles **unless** a location (index) for the role is specified. To "reorder" roles, you must either:

- revoke the roles and grant them in the desired order;
- use the `--index` option to specify the location to insert the role;
- use *setRoles(idens)* to replace the user's roles with a new list of roles; or
- use the commercial Synapse UI (Optic) to reorder the roles using drag-and drop.

See *Permissions Background* for additional detail on permissions and *Precedence*.

4.3 Assign and Manage Permissions

Synapse provides a highly flexible system of role-based access control (RBAC). **Rules** are used to assign permissions to users and / or roles, with a defined order of precedence for how permissions are evaluated.

Permissions can be assigned very broadly, such as allowing a user (or role) to create / modify / delete any node. Permissions can also be very fine-grained, restricting users so that they can **only** create specific nodes, set specific properties, create specific edges, or apply specific tags.

4.3.1 Permissions Background

Before describing how to assign and manage permissions in Synapse, it is helpful to define some key components of Synapse and the permissions ecosystem.

Services

Synapse is designed as a modular set of **services.** A service can be thought of as a container used to run an application. **Synapse services** make up the core Synapse architecture, and include the *Cortex* (data store), *Axon* (file storage), and the commercial *Optic* UI. Services handle user authentication and authorization.

From a Synapse Admin perspective, you will primarily be concerned with managing user accounts and permissions to (and within) the Synapse **Cortex.**

Tip: When we talk about "Synapse users" or "permissions to Synapse" we are generally referring to user accounts and roles in a Cortex, and permissions to a Cortex and its associated objects.

Depending on your Synapse deployment, you may need to grant or manage permissions to additional Synapse services. See the sections on *Optic Permissions* and *Power-Up Permissions* for details.

Cortex

The **Cortex** is Synapse's primary data store. Users and roles are created and managed in the Cortex, and most things for which users will need permissions apply to the Cortex and to the views, layers, and data (nodes, tags, etc.) that reside there.

Auth Gate

An **Auth Gate** (or "gate", informally) is an object within a service (such as a Cortex) that may have its own set of permissions. A *View* and a *Layer* are both common examples of Auth Gates.

Auth Gates are represented by a 128-bit identifier (iden) that uniquely identifies the Auth Gate object itself. They also have an associated type to specify the kind of Auth Gate object (e.g., "view"). Some Auth Gates also support the use of "user friendly" names, though this is dependent on the type of Auth Gate and has no impact on the underlying iden or associated permissions.

Scope

Scope refers to the object to which a particular permission applies. For example, permissions granted on an Auth Gate (such as a view) are scoped to (or **local** to) that Auth Gate. Permissions granted at the Cortex level are **global** with respect to the Cortex.

Scope affects the order (precedence) in which permissions are evaluated.

Permission

A **permission** is a string that is used to control access. For example:

`view.add`

Tip: A list of most permissions available in a Cortex can be found under *Cortex Permissions*. You can also display the list in Synapse using the `auth.list.perms` command.

Most permission strings use a dotted (hierarchical) format; specifying a permission higher up in the hierarchy includes all permissions below it. For example, the permission `view` includes all of the following permissions: `view.add`, `view.del`, `view.read`, and `view.set`.

Permissions related to objects such as nodes or tags can optionally extend the permission string to be highly specific, referencing particular forms, properties, tags/tag trees, or light edges. This allows you to set highly granular permissions.

Granular permissions may be useful for organizations with specialized users or teams, where certain individuals are responsible for specific types of analysis (e.g., strategic analysis vs. tactical threat tracking) and should be the only users authorized to create, modify, and tag certain types of data.

Granular permissions can also be used to differentiate between senior and junior roles; for example, only senior analysts may be allowed to apply tags representing certain assessments (such as attribution).

Examples:

Description	Permission
Perform **any** action on **any** kind of node (including deleting nodes and working with properties, tags, edges, and node data)	`node`
Add any kind of node (but not delete nodes, or work with properties, tags, edges, or node data)	`node.add`
Only add `inet:ipv4` nodes (but not set properties, or work with tags or edges)	`node.add.inet:ipv4`
Only add (set) the `:asn` property of `inet:ipv4` nodes (but not create nodes or work with other properties, tags, edges, etc.)	`node.prop.set.inet:ipv4:asn`
Add or remove **any** tag (Note that adding/removing tags may require the ability to create `syn:tag` nodes, unless those nodes already exist.)	`node.tag`
Only add and remove tags in the "mytag" tag tree	`node.tag.add.mytag` `node.tag.del.mytag`
Add or remove **any** edge (Note that adding or removing edges allows creating edges between **any** nodes; there are no model constraints on the kinds of nodes that can be joined. It also allows the creation of new / arbitrarily named edges.)	`node.edge`
Only add edges	`node.edge.add`
Only add `refs` edges	`node.edge.add.refs`

Note: Permissions strings **do not** support wildcards (`*`). For example, you cannot specify `node.tag.*.mytag` to allow users to both add and delete tags in the `mytag` tree.

Rule

A **rule** is used to grant (or prohibit) a specific permission. Rules are evaluated in a defined order of precedence.

When you specify a rule, there is an implicit **allow** directive; a permission string by itself indicates the permission is allowed/true:

`view.add`

To use a rule to **deny** a permission, use the "not" or "bang" symbol (`!`) to indicate the permission is denied/false:

`!node.tag.add.mytag`

Precedence

Rules in Synapse are evaluated in order of **precedence.** A requested action will be allowed (or denied) based on the **first matching rule** found for the action. If no matching rule is found, the action is **denied.**

Generally speaking, rules are evaluated from "most specific" to "least specific". Rules are evaluated in the following order:

- **User** rules at the **local** (i.e., Auth Gate) level.
- **Role** rules at the **local** level.
- **User** rules at the **global** (i.e., Cortex) level.
- **Role** rules at the **global** level.

Note: Because global rules are evaluated after local rules, permissions granted at the global level can "override" permissions that are not explicitly denied at the local level. For example, a user may fork a view (making them admin of that view) and grant "read" access to a coworker (`view.read`).

If the coworker has "write" permissions (such as `node.tag`) at the **global** level, they will be able to add tags within the forked view (or any view where they have `view.read` permissions).

If the user forking the view also specified `!node` for the view's layer, the coworker would be prevented from adding any tags in the forked view (or making any edits whatsoever).

Roles (granted to a user) and **rules** (assigned to a user or role) are **also ordered:**

- When granting roles to a user, each new role is added to the **end** of the list of roles **unless** a location (index) for the role is specified.
- When assigning rules to a role or user, each new rule is added to the **end** of the list of rules **unless** a location (index) for the rule is specified.

Rules and roles are evaluated in the following order:

- **User rules** are evaluated in order from first to last.
- Each **role** granted to a user is evaluated in order from first to last.
- For each role, the **role's rules** are evaluated in order from first to last.

This means that the same rules, applied and evaluated in a different order, will give different results. As a simple example:

These rules will **allow** the creation of `file:bytes` nodes, but no other nodes:

```
node.add.file:bytes
!node.add
```

The same rules in the opposite order will **disallow** the creation of **any** nodes:

```
!node.add
node.add.file:bytes
```

Admin

Admin status allows a user to **bypass all permissions checks** for the **scope** where the user is admin. For example, a Synapse (Cortex) admin user can bypass all Cortex permissions checks (can "do anything" within the Cortex).

Users are generally **admin** of objects that they create. A user who forks a view is **admin** for the view that they fork, and can bypass all permissions checks ("do anything") within the forked view.

Note: It is not possible to assign **admin** privileges to a role.

Easy Permissions

Easy permissions ("easy perms" for short) is a mechanism that simplifies granting common sets of permissions to users or roles for a particular object. Where easy perms are used, you can specify four levels of access: **deny, read, edit,** and **admin.** These access levels have corresponding integer values:

- Deny = 0
- Read = 1
- Edit = 2
- Admin = 3

Easy perms apply to specific objects. Where easy perms are available, the following conventions apply:

- The user who creates the object has **admin** privileges for that object.
- **Admin** privileges include the ability to grant permissions to others (including the ability to explicitly deny access).
- Admin privileges are required to **delete** the object (i.e., **edit** permissions do not include **delete**).

Tip: $lib.macro.grant library is an example of where easy permissions can be used to assign permissions.

Views and Layers

Data in a Cortex is stored in one or more **layers** (see *Layer*). Layers are composed into **views** (see *View*) containing the data that should be visible to users or roles. (A standard installation of Synapse consists of the default view, which contains one layer.)

Views define the data that a user or role can **see** - they act as a **read** boundary. Granting the `view.read` permission on a view allows users to see (read) data in any of the view's layers; you do not need to explicitly grant "read" access to the individual layers themselves.

The ability to read data in a view is "all or nothing" - you cannot allow users to see some nodes in a view but not others. (Sensitive data should be stored in its own layer, and views containing that layer should be limited to users or roles with a need to access that data.)

Layers define the changes (if any) that a user or role can make to data in Synapse - they act as a **write** boundary. In normal circumstances, only the top layer in a view is writable. The ability to write data **to** a layer is controlled by the various `node.*` permissions, which specify the forms / properties / tags / light edges a user or role can work with (create / modify / delete). Permissions to modify data must be assigned at the appropriate **layer** (or globally, if the permissions apply to all writable layers in the Cortex).

4.3.2 Assign Permissions

You assign (allow or deny) permissions in Synapse by adding rules to (or removing rules from) roles or users. Recall that **order matters** when adding rules (see *Precedence*).

From a Synapse Admin perspective, managing permissions within Synapse commonly involves:

- Assigning rules to users and roles within the Cortex.
- Assigning rules to users and roles for various Auth Gates (such as layers or views) if necessary.
- Assigning rules to users and roles to allow or deny access to additional services, such as various Power-Ups.

Permissions in Synapse are managed using the Storm *auth* commands.

In the commercial Optic UI, permissions can also be managed through the **Admin Tool** and through dialogs associated with various objects (such as Views or Stories).

Tip: If a user attempts an action that they do not have permissions to perform, Synapse will return an `AuthDeny` error that lists the specific permission that is required.

Note: The descriptions and examples below assume that you have deployed Synapse using native Synapse management and authentication of users, roles, and permissions.

Default Permissions

Synapse includes the following default permissions:

- The built-in **root** user has **admin** access (`admin=true`) to the Cortex.
- The built-in **all** role has **read** access (`view.read`) to any view created with `worldreadable=True`. This includes the **default** view.

Any additional permissions must be **explicitly granted** to users or roles. In all but a few edge cases, Synapse assumes an implicit default `deny all` as the final rule evaluated when checking permissions.

Note: There are a few edge cases where a specific permission assumes a **default allow** instead of a **default deny,** but these are uncommon. These cases are highly specific, and usually arise in cases where a **new** permission has been implemented. That is, an action that was not originally subject to a permissions check now has one (usually because of a need to explicitly **deny** that action to particular users or roles).

If a previously unchecked action were added with "default deny", it would potentially break existing Synapse deployments by suddenly blocking an action that had been previously allowed (ungated). In these circumstances the new permission is given a "default allow" that can then be specifically denied if necessary.

Available Permissions

The *auth.perms.list* command can be used to display the set of permissions available in your Cortex. This includes native Synapse permissions as well as any permissions associated with other packages and services.

Sample output for this command can be seen under *Cortex Permissions*. The permissions available on your Cortex may vary depending on the services and packages installed (e.g., such as Power-Ups).

Global (Cortex) Permissions

Permissions in Synapse can be assigned at the global (Cortex) level, or to a specific Auth Gate (see *Auth Gate Permissions*). To assign permissions to an Auth Gate, you must specify its identifier (iden) (i.e., using the `--gate` option to the appropriate Storm command) when adding the associated rule to a user or role.

If you do not specify an Auth Gate, the permissions are **global** and apply to any / all instances within the Cortex where a user or role has access. For example, the following Storm command:

```
auth.role.addrule all node
```

…grants (allows) the `node` permission to the built-in **all** role. This allows **any** user (because all users are granted the **all** role by default) to perform **any** action on **any** node in **any** layer that is the topmost (writeable) layer in **any** view that the user can see.

Specifying rules at the global (Cortex) level may be sufficient for many basic Synapse deployments.

Note: Recall that **order matters** when adding rules:

- by default, each rule is added to the **end** of the list of rules assigned to a user or role; and
- rules are evaluated in order of precedence.

To reorder rules, you must:

- use the `--index` option with `auth.user.addrule` or `auth.role.addrule` to specify a location to insert a specific rule;
- remove and re-add the rules in the desired order;
- use *setRules(rules, gateiden=$lib.null)* or *setRules(rules, gateiden=$lib.null)* to replace the rules for a user or role with a new set of rules; or
- use the commercial Synapse UI (Optic) to reorder rules using drag-and-drop.

Assign Permissions

Permissions rules (allow or deny) are assigned using the *auth.user.addrule* and *auth.role.addrule* commands.

Examples:

Prevent the user "ron" from setting tag descriptions (setting the `syn:tag:desc` property):

```
storm> auth.user.addrule ron "!node.prop.set.syn:tag:desc"
Added rule !node.prop.set.syn:tag:desc to user ron.
```

Tip: Deny rules specified with Storm must be enclosed in quotes (single or double) because they begin with a symbol (!).

Allow the role "senior analysts" to add tags in threat attribution (`cno.threat`) tag tree:

```
storm> auth.role.addrule "senior analysts" node.tag.add.cno.threat
Added rule node.tag.add.cno.threat to role senior analysts.
```

Prevent the "all" role from deleting nodes:

```
storm> auth.role.addrule all "!node.del"
Added rule !node.del to role all.
```

Prevent the "all" role from deleting nodes, and insert this as the first rule for the role:

```
storm> auth.role.addrule --index 0 all "!node.del"
Added rule !node.del to role all.
```

Tip: Recall that you can *Display a User* or *Display a Role* with the *auth.user.show* and *auth.role.show* commands.

Revoke Permissions

Permissions rules are revoked using the `auth.user.delrule` and `auth.role.delrule` commands.

Examples:

Revoke the rule that prevents user "ron" from setting tag descriptions:

```
storm> auth.user.delrule ron "!node.prop.set.syn:tag:desc"
Removed rule !node.prop.set.syn:tag:desc from user ron.
```

Revoke the rule that allows "junior analysts" to apply tags in the `cno.threat` tag tree:

```
storm> auth.role.delrule "junior analysts" node.tag.cno.threat
Removed rule node.tag.cno.threat from role junior analysts.
```

Check Permissions

The *auth.user.allowed* command can be used to check whether a user has a particular permission (i.e., is allowed to perform the associated operation) for a specific **scope** (i.e., globally or for an individual Auth Gate). If an appropriate `allow` rule exists, the command will show the source (i.e., the rule, role, and / or associated Auth Gate) where the permission has been assigned.

Tip:

- A user may have permissions locally (e.g., to a specific Auth Gate) that they do not have globally. In other words a **global** check may (correctly) show that a user does **not** have an expected permission globally, but the permission will show as "allowed" when the appropriate Auth Gate is checked.
- When checking whether a user can see (read) data or manipulate (e.g., fork) a view, check the relevant **view.**
- When checking whether a user can modify (write or delete) data, check the relevant **layer.**

Examples:

Check whether user 'ron' is allowed to apply tags in the `cno` tag tree **globally:**

```
storm> auth.user.allowed ron node.tag.add.cno
allowed: true - Matched role rule (node.tag.add.cno) for role cattribution analyst.
```

Check whether user 'ron' is allowed to apply tags in the `cno` tag tree in the **current layer:**

```
storm> auth.user.allowed --gate $lib.layer.get().iden ron node.tag.add.cno
allowed: true - Matched role rule (node.tag.add.cno) for role cattribution analyst.
```

Note that the response for each of the commands above is identical, even though the first example performed a global check (no `--gate` option) while the second example checked the current layer (retrieved with `$lib.layer.get()`). The response in the second example shows that Ron can apply tags in the current layer because he has **global** permissions for this action - indicated by the **absence** of an iden in the response. If Ron's permissions were restricted to the queried gate (in this case, the layer), the associated iden would have been included in the command output.

Check whether user 'ron' is allowed to fork the **current view:**

```
storm> auth.user.allowed --gate $lib.view.get().iden ron view.add
allowed: false - No matching rule found.
```

Auth Gate Permissions

To assign permissions for an Auth Gate, you use the same Storm commands used to assign global permissions, but you must specify the Auth Gate's full identifier (iden) (using the `--gate` option) when adding or removing the rule.

Obtain a Gate's Iden

The Storm *view* and *layer* commands can be used to manage views and layers, respectively. In particular, the following commands are useful for displaying all views or layers (including their idens), or displaying a specific view or layer:

- *view.list*
- *view.get*
- *layer.list*
- *layer.get*

Examples:

Display all views:

```
storm> view.list

View: e97aedae71082cd66a877fe3a3f08c02 (name: default)
  Creator: b64fceab1e4e01cbd3ab0645a2e49105
  Layers:
    1b0a73c2fda8f48bc24d5962a9351821: default readonly: False
```

Display the current layer:

```
storm> layer.get
Layer: 1b0a73c2fda8f48bc24d5962a9351821 (name: default) readonly: False creator:␣
↪b64fceab1e4e01cbd3ab0645a2e49105
```

View a Gate's Permissions

The `auth.gate.show` command is used to display permissions information about a particular Auth Gate (e.g., a view or layer). You can provide the specific iden for an Auth Gate, or use the syntax below to retrieve information for the **current** view or layer. (Viewing information for the "current layer" will return information for the top layer of the current view.)

Example:

Display information for the current view:

```
storm> auth.gate.show $lib.view.get().iden
Gate Type: view

Auth Gate Users:
  b64fceab1e4e01cbd3ab0645a2e49105 - root
    Admin: true
    Rules:

Auth Gate Roles:
  b07861cadfc09ee2d29bba1b4240e333 - all
    Rules:
      [0  ] - view.read
```

Display information for the current layer (i.e., the top layer of the current view):

```
storm> auth.gate.show $lib.layer.get().iden
Gate Type: layer

Auth Gate Users:
  b64fceab1e4e01cbd3ab0645a2e49105 - root
    Admin: true
    Rules:

Auth Gate Roles:
  b07861cadfc09ee2d29bba1b4240e333 - all
    Rules:
      [0  ] - layer.read
```

4.3.3 Permissions Best Practices

- Synapse Admins should use a designated admin account for administrative tasks and a separate account for their user tasks.
- Where possible, assign permissions to roles and grant roles to users vs. assigning permissions to users directly.
- Create a general purpose role (such as `users`, or use the built-in `all` role) and assign the basic permissions that **all** Synapse users should have to this role. This includes "things all users should be able to do" (allow rules) as well as "things all users should be **explicitly** prohibited from doing" (deny rules). Create additional roles as needed to allow (or further restrict) specific operations.
- Segregate data with different access requirements into different **layers.** Grant access to data sets by composing those layers into **views** and granting roles access to the appropriate view(s).
- The ability to **delete nodes** in Synapse should be granted to a limited number of trusted individuals. We recommend creating a dedicated role for this purpose.
- If a role will have **limited permissions,** it is generally easier to **explicitly allow** only those actions; everything else will be denied by default.
- If a role or user will have **broad permissions** with some restrictions, it is generally easier to **explicitly deny** the restricted actions first, and then **grant** broad permissions (for example `!node.del` followed by `node`). Because permissions rules are checked in order, Synapse will encounter any deny rules first (i.e., user is unable to delete nodes), blocking the prohibited action while then allowing anything not specifically denied (i.e., user can do anything else to nodes).

4.3.4 Example Permissions

The examples below illustrate a few common use cases for roles and permissions within Synapse. These rule sets are meant as simple illustrations and do not necessarily illustrate fully-defined, production-ready permission sets.

Recall that:

- Views control **read** access to the data store. Users with read access to a view (`view.read`) can read all data in all layers of the view (i.e., no additional layer-specific permissions are required for read access).
- Layers control **write** access to the data store. Use permissions to manage the data that can be written to a given layer (including the ability to merge data into that layer from a forked view).
- A user who can fork a view is **admin** within their forked view.

A list of available Cortex permissions is available under the *Cortex Permissions* section, or can be viewed in Synapse with the `auth.perms.list` command.

Case 1 - Grant common permissions - basic

These basic permissions can be assigned to a role to allow users to perform common operations in Synapse.

Permission	Description
`view.read`	See / read any view
`view.add`	Fork any view they can see
`node`	Create, modify, or delete any type of data (nodes, properties, light edges, tags, and node data) in the top layer of any view they can see

Tips:

- The `all` role has implicit (and non-revocable) "read" access to Synapse's **default** view. This is not the same as global `view.read` access. To allow the `all` role (or any role) to see other views, you must explicitly assign the `view.read` permission (either globally or to individual views).
- Users can only fork (`view.add`) views they can see (`view.read`). If users should be allowed to fork any view where they have read access, the `view.add` permission can be assigned **globally** even if read access is managed on a **per-view** basis.

Case 2 - Grant common permissions - intermediate

These permissions expand on Case 1, but only allow the role to see **specific** views (by granting `view.read` locally to individual views).

Any **global** permissions (e.g., `node.add`) will apply to the top (writeable) layer of **any** view the role can see, unless the permissions are overridden locally.

These permissions also prevent the role from **deleting nodes** globally, while allowing them to delete properties or edges and to remove tags.

Permission	Scope	Description
`view.add`	global	Fork any view they can see (based on `view.read`)
`!node.del`	global	Prevent deletion of any nodes
`node.add`	global	Create nodes in the top layer of any view they can see
`node.prop`	global	Set, modify, or delete node properties in the top layer of any view they can see
`node.edge`	global	Add or remove light edges in the top layer of any view they can see
`node.tag`	global	Add or remove tags from nodes in the top layer of any view they can see
`view.read`	local	See all the data in all the layers of the specific view(s) where the rule is assigned

Case 3 - Create a dedicated role that can delete nodes

Deleting nodes indiscriminately or incorrectly can negatively impact your data store (i.e., leaving "holes" in the graph or destroying data). Synapse requires that users run an explicit command (*delnode*) to delete nodes, so the action is a deliberate choice (vs. an "accidental click").

We strongly recommend that you create a role whose sole permission is the ability to delete nodes, and grant that role to a limited number of users. To do this:

- **Explicitly deny** permission to delete nodes (`!node.del`) at the **global** level to the general purpose role you use to manage permissions for all users (as shown in Case 2 above).
- Create a dedicated role whose only permission will be the ability to delete nodes.
  - We encourage a name that inspires caution, such as `fire ze missiles` or `agents of destruction`, but you can just use `deleters`.
- Assign the `node.del` rule to the role (globally, or for specific layers).

Tips:

- All delete operations (whether deleting nodes, properties, edges, or removing tags) must be performed directly in the layer where the data resides. As admin of any view that they fork, "normal" users can delete data created or modified **within** their forked view.

Case 4 - Place guardrails around writing (creating or merging) data

Permissions can be used to prevent roles from:

- creating various types of data directly in a layer/view; or
- merging various types of data into an underlying view (technically, to the view's top layer).

These types of permissions can help ensure that data in a "production" layer remains as pristine and error-free as possible. For example:

- Help to limit typos that result in "bad" tags or edges.
- Prevent data from a sensitive or restricted layer from being written to a non-restricted layer.

Example use cases:

- Use permissions around light edges to only allow the creation of specific named edges. This can limit typos in edge names and/or prevent users from creating arbitrarily named edges.
- Use permissions around tags or tag trees to only allow applying certain tags (e.g., to enforce your organization's tag conventions). For example, permissions can ensure that users' "scratch" tags (`#thesilence.mywork`) or tags indicating sensitive data (`#tlp.red`) are not added to "production" data.
- Use permissions around individual properties to prohibit setting specific properties in particular layers. For example, taxonomy properties (such as `risk:threat:type`) may be "under development" in an internal analysis view while users test and agree on appropriate categories. You may want to prevent this property from being set (merged) into production data until the taxonomy is finalized.

Tip: The degree to which you enforce data and tag conventions through permissions vs. by consensus (i.e., users agree on "best efforts" to keep the data tidy) will depend on your organization and your use case. Managing permissions adds overhead, but may be worth the effort for data sets that require high fidelity or quality. The overhead may have less benefit for internal data or test data where occasional errors have minimal impact and can be "cleaned up" as needed.

The sample rules below can be applied **globally** (a user with this role can write "approved" data to any writeable layer of any view they can see) or **locally** to specific layers.

The examples below **only** illustrate how certain write actions can be restricted and do not address other permissions that a user/role might need. These permissions could be added to an existing role (such as your general `users` role), or granted via their own role.

Example 1:

If a limited set of actions are allowed, simply specify the changes that the role can make. Anything else is implicitly denied by default.

In this example, a role with the following permissions can:

- **only** add and remove tags in the listed tag trees; and
- **only** create and delete the listed edges.

Permission	Description
`node.tag.add.cno`	Add / apply tags in the `cno` tree (e.g., `cno`, `cno.mal`, `cno.mal.plugx` etc.)
`node.tag.del.cno`	Remove any tags in the `cno` tree
`node.tag.add.rep`	Add / apply any tags in the `rep` tree
`node.tag.del.rep`	Remove any tags in the `rep` tree
`node.edge.add.refs`	Add `refs` light edges
`node.edge.del.refs`	Delete `refs` light edges
`node.edge.add.uses`	Add `uses` light edges
`node.edge.del.uses`	Delete `uses` light edges
`node.edge.add.targets`	Add `targets` light edges
`node.edge.del.targets`	Delete `targets` light edges

Example 2:

If specific actions are prohibited, **deny** those changes and then **allow** "everything else".

A role with the following permissions is **prohibited** from:

- Creating `risk:threat:type:taxonomy` nodes (representing "categories" of threats).
- Setting the `:type` property for `risk:threat` nodes (e.g., specifying the taxonomy category for a particular threat).
- Creating tags in the `tlp` tree.

Note that the permissions as listed only prohibit actions. For a role with these permissions to be able to make other changes (e.g., add other nodes or edges), those permissions need to be granted after these "deny" rules, or as part of another role.

Permission	Description
`!node.add.risk:threat:type:taxonomy`	Prevent creating these nodes
`!node.prop.set.risk:threat:type`	Prevent setting this property (i.e., on existing `risk:threat` nodes)
`!node.tag.add.tlp`	Prevent applying tags in the `tlp` tree

Tip: To prevent users or roles from making **any** changes to a particular view (i.e., users cannot merge any data into the view / write any data directly to the view's topmost layer):

- Do not add `node` permissions to the view's topmost layer (write permissions that are not granted are implicitly denied).
- If a role has been granted `node` (or similar) permissions **globally,** override this by explicitly denying (`!node`) the permission on the layer you want to protect.

Case 5 - Senior vs. junior roles

Senior roles (with more permissions) and junior roles (with limited permissions) are used in a variety of situations, such as new trainees vs. experienced users or junior vs. senior analysts.

When using a "fork and merge" workflow, a junior user can "do anything" (as **admin**) in a view that they fork. This allows them to enrich data and annotate their assessments using tags. But permissions can prevent them from merging some (or all) data and tags until a senior user has reviewed the changes. The senior role (with appropriate permissions) can then merge the data on the junior user's behalf.

For example, tags representing key analytical assessments - such as determining if a file or indicator is associated with a malware family, or tags representing threat clustering and attribution - may require careful consideration. The ability to merge these tags may be limited to senior analysts who can verify that the junior analyst has applied them correctly.

These types of permissions are typically **cumulative;** generic users may be prohibited (or simply not allowed) to perform a certain action, with additional permissions granted to increasingly senior or experienced roles. In the example below, all users would have the general `users` role and analysts would be granted **each** additional role as they gained experience.

Example:

Role	Permission(s)	Description

users	`!node.tag.cno`
	`!node.tag.rep`
	`node.tag`
		Prevent applying tags in the `cno` and `rep` trees (representing specific analytical assessements) but apply other tags

novice analyst	`node.tag.add.rep`	Novices can apply tags in the `rep` tree (representing third-party reporting)
junior analyst	`node.tag.add.cno.infra`	Junior analysts can apply tags in the `cno.infra` tree (related to network infrastructure)
senior analyst	`node.tag.add.cno.threat` `node.tag.add.cno.mal`	Senior analysts can apply tags in the `cno.threat` and `cno.mal` trees (assessments related to threat clusters and malware families)

Note: Because of *Precedence*, as additional roles are granted, they would need to be added (indexed) **before** the `users` role to prevent that role's explicit deny permissions from overriding the newly allowed tag privileges.

Case 6 - Specialized roles

For organizations with diverse analysis teams (e.g., where analysts specialize in particular areas) or organizations where multiple teams or departments use Synapse for different purposes, it may be helpful to create highly specialized roles.

Examples:

- An organization with a dedicated malware analysis team may **only** allow those specialists to apply tags related to malware/code families and malware ecosystems.
- An organization's strategic analysts may be solely responsible for certain objects and related data. For example, a strategic team may be in charge of researching and creating organizations (`ou:org`) and associated industries (`ou:industry`) in order to track victimology. Strategic analysts can ensure that these objects are created according to the team's standards and that organizations are assigned to the appropriate industries.

Malware analyst example:

- We assume the ability to apply the specialized tags listed below is either **not granted** or **explicitly denied** elsewhere/to other roles.
- Malware analysts can also be granted the ability to **remove** the tags listed below with the corresponding `node.tag.del` permissions.

Permission	Description
`node.tag.add.cno.code`	Apply `cno.code` tags (designating specific samples of code families - e.g., `cno.code.plugx`)
`node.tag.add.cno.mal`	Apply `cno.mal` tags (designating components of malware / code family ecosystems, such as related droppers or C2 - e.g., `cno.mal.plugx`)
`node.tag.add.cno.rel`	Apply `cno.rel` tags (designating components that may be observed as part of a malware ecosystem but are not inherently malicious - e.g., `cno.rel.plugx`)

Strategic analyst example:

- We assume the ability to create these nodes / set these properties is either **not granted** or **explicitly denied** elsewhere/to other roles.
- Strategic analysts can optionally be granted the ability to delete relevant nodes/properties with the corresponding `node.del` or `node.prop.del` permissions.
- Depending on how you assign permissions, keep in mind that roles that cannot **create** nodes may still be able to **set or modify properties** on the node as long as the node already exists. This ability can be restricted via additional `node.prop.set` rules if necessary.

Permission	Description
`node.add.ou:org`	Create organization nodes
`node.prop.set.ou:org:industries`	Assign organizations to one or more industries
`node.add.ou:industry`	Create industry nodes

4.3.5 Cortex Permissions

The following is a list of the Cortex permissions that may be granted to a user or role. If a gate other than `cortex` is specified, the permission will be checked against the specific gate instance and if no match is found, it will be checked against the global rules.

```
storm> auth.perms.list
auth.role.set.name
    Permits a user to change the name of a role.
    gate: cortex
    default: false

auth.role.set.rules
    Permits a user to modify rules of a role.
    gate: cortex
    default: false

auth.self.set.apikey
    Permits a user to manage their API keys.
    gate: cortex
    default: true

auth.self.set.email
    Permits a user to change their own email address.
    gate: cortex
    default: true

auth.self.set.name
    Permits a user to change their own username.
    gate: cortex
    default: true

auth.self.set.passwd
    Permits a user to change their own password.
    gate: cortex
    default: true

auth.user.get.profile.<name>
    Permits a user to retrieve their profile information.
    gate: cortex
    default: false
    example: auth.user.get.profile.fullname

auth.user.grant
    Controls granting roles to a user.
    gate: cortex
    default: false

auth.user.pop.profile.<name>
    Permits a user to remove profile information.
    gate: cortex
    default: false
    example: auth.user.pop.profile.fullname
```

(continues on next page)

(continued from previous page)

```
auth.user.revoke
    Controls revoking roles from a user.
    gate: cortex
    default: false

auth.user.set.admin
    Controls setting/removing a user's admin status.
    gate: cortex
    default: false

auth.user.set.apikey
    Permits a user to manage API keys for other users. USE WITH CAUTUON!
    gate: cortex
    default: false

auth.user.set.email
    Controls changing a user's email address.
    gate: cortex
    default: false

auth.user.set.locked
    Controls locking/unlocking a user account.
    gate: cortex
    default: false

auth.user.set.passwd
    Controls changing a user password.
    gate: cortex
    default: false

auth.user.set.profile.<name>
    Permits a user to set profile information.
    gate: cortex
    default: false
    example: auth.user.set.profile.fullname

auth.user.set.rules
    Controls adding rules to a user.
    gate: cortex
    default: false

axon.del
    Controls the ability to remove a file from the Axon.
    gate: cortex
    default: false

axon.get
    Controls the ability to retrieve a file from the Axon.
    gate: cortex
    default: false
```

(continues on next page)

(continued from previous page)

```
axon.has
    Controls the ability to check if the Axon contains a file.
    gate: cortex
    default: false

axon.upload
    Controls the ability to upload a file to the Axon.
    gate: cortex
    default: false

backup.del
    Permits a user to delete an existing backup.
    gate: cortex
    default: false

backup.list
    Permits a user to list existing backups.
    gate: cortex
    default: false

backup.run
    Permits a user to create a backup.
    gate: cortex
    default: false

cron.add
    Permits a user to create a cron job.
    gate: view
    default: false

cron.del
    Permits a user to remove a cron job.
    gate: cronjob
    default: false

cron.get
    Permits a user to list cron jobs.
    gate: cronjob
    default: false

cron.set
    Permits a user to modify/move a cron job.
    gate: cronjob
    default: false

cron.set.creator
    Permits a user to modify the creator property of a cron job.
    gate: cortex
    default: false

globals
    Used to control all operations for global variables.
```

(continues on next page)

(continued from previous page)

```
    gate: cortex
    default: false

globals.get
    Used to control read access to all global variables.
    gate: cortex
    default: false

globals.get.<name>
    Used to control read access to a specific global variable.
    gate: cortex
    default: false

globals.pop
    Used to control delete access to all global variables.
    gate: cortex
    default: false

globals.pop.<name>
    Used to control delete access to a specific global variable.
    gate: cortex
    default: false

globals.set
    Used to control edit access to all global variables.
    gate: cortex
    default: false

globals.set.<name>
    Used to control edit access to a specific global variable.
    gate: cortex
    default: false

model.form.add
    Controls access to adding extended model forms.
    gate: cortex
    default: false

model.form.add.<form>
    Controls access to adding specific extended model forms.
    gate: cortex
    default: false
    example: model.form.add._foo:bar

model.form.del
    Controls access to deleting extended model forms.
    gate: cortex
    default: false

model.form.del.<form>
    Controls access to deleting specific extended model forms.
    gate: cortex
```

(continues on next page)

(continued from previous page)

```
    default: false
    example: model.form.del._foo:bar

model.prop.add
    Controls access to adding extended model properties.
    gate: cortex
    default: false

model.prop.add.<form>
    Controls access to adding specific extended model properties.
    gate: cortex
    default: false
    example: model.prop.add._foo:bar

model.prop.del
    Controls access to deleting extended model properties.
    gate: cortex
    default: false

model.prop.del.<form>
    Controls access to deleting specific extended model properties.
    gate: cortex
    default: false
    example: model.prop.del._foo:bar

model.tagprop.add
    Controls access to adding extended model tag properties.
    gate: cortex
    default: false

model.tagprop.del
    Controls access to deleting extended model tag properties.
    gate: cortex
    default: false

model.univ.add
    Controls access to adding extended model universal properties.
    gate: cortex
    default: false

model.univ.del
    Controls access to deleting extended model universal properties.
    gate: cortex
    default: false

node
    Controls all node edits in a layer.
    gate: layer
    default: false

node.add
    Controls adding any form of node in a layer.
```

(continues on next page)

(continued from previous page)

```
    gate: layer
    default: false

node.add.<form>
    Controls adding a specific form of node in a layer.
    gate: layer
    default: false
    example: node.add.inet:ipv4

node.del
    Controls removing any form of node in a layer.
    gate: layer
    default: false

node.del.<form>
    Controls removing a specific form of node in a layer.
    gate: layer
    default: false

node.prop
    Controls editing any prop on any node in the layer.
    gate: layer
    default: false

node.prop.del
    Controls removing any prop on any node in a layer.
    gate: layer
    default: false

node.prop.del.<form>
    Controls removing any property from a form of node in a layer.
    gate: layer
    default: false
    example: node.prop.del.inet:ipv4

node.prop.del.<form>.<prop>
    Controls removing a specific property from a form of node in a layer.
    gate: layer
    default: false
    example: node.prop.del.inet:ipv4.asn

node.prop.set
    Controls setting any prop on any node in a layer.
    gate: layer
    default: false

node.prop.set.<form>
    Controls setting any property on a form of node in a layer.
    gate: layer
    default: false
    example: node.prop.set.inet:ipv4
```

(continues on next page)

(continued from previous page)

```
node.prop.set.<form>.<prop>
    Controls setting a specific property on a form of node in a layer.
    gate: layer
    default: false
    example: node.prop.set.inet:ipv4.asn

node.tag
    Controls editing any tag on any node in a layer.
    gate: layer
    default: false

node.tag.add
    Controls adding any tag on any node in a layer.
    gate: layer
    default: false

node.tag.add.<tag...>
    Controls adding a specific tag on any node in a layer.
    gate: layer
    default: false
    example: node.tag.add.cno.mal.redtree

node.tag.del
    Controls removing any tag on any node in a layer.
    gate: layer
    default: false

node.tag.del.<tag...>
    Controls removing a specific tag on any node in a layer.
    gate: layer
    default: false
    example: node.tag.del.cno.mal.redtree

pkg.add
    Controls access to adding storm packages.
    gate: cortex
    default: false

pkg.del
    Controls access to deleting storm packages.
    gate: cortex
    default: false

service.add
    Controls the ability to add a Storm Service to the Cortex.
    gate: cortex
    default: false

service.del
    Controls the ability to delete a Storm Service from the Cortex
    gate: cortex
    default: false
```

(continues on next page)

(continued from previous page)

```
service.get
    Controls the ability to get the Service object for any Storm Service.
    gate: cortex
    default: false

service.get.<iden>
    Controls the ability to get the Service object for a Storm Service by iden.
    gate: cortex
    default: false

service.list
    Controls the ability to list all available Storm Services and their service␣
↪definitions.
    gate: cortex
    default: false

storm.asroot.cmd.<cmdname>
    Controls running storm commands requiring root privileges.
    gate: cortex
    default: false
    example: storm.asroot.cmd.movetag

storm.asroot.mod.<modname>
    Controls importing modules requiring root privileges.
    gate: cortex
    default: false
    example: storm.asroot.cmd.synapse-misp.privsep

storm.graph.add
    Controls access to add a storm graph.
    gate: cortex
    default: true

storm.inet.imap.connect
    Controls connecting to external servers via imap.
    gate: cortex
    default: false

storm.inet.smtp.send
    Controls sending SMTP messages to external servers.
    gate: cortex
    default: false

storm.lib.auth.roles.add
    Controls the ability to add a role to the system. USE WITH CAUTION!
    gate: cortex
    default: false

storm.lib.auth.roles.del
    Controls the ability to remove a role from the system. USE WITH CAUTION!
    gate: cortex
```

(continues on next page)

(continued from previous page)

```
    default: false

storm.lib.auth.users.add
    Controls the ability to add a user to the system. USE WITH CAUTION!
    gate: cortex
    default: false

storm.lib.auth.users.del
    Controls the ability to remove a user from the system. USE WITH CAUTION!
    gate: cortex
    default: false

storm.lib.axon.del
    Controls the ability to remove a file from the Axon.
    gate: cortex
    default: false

storm.lib.axon.get
    Controls the ability to retrieve a file from the Axon.
    gate: cortex
    default: false

storm.lib.axon.has
    Controls the ability to check if the Axon contains a file.
    gate: cortex
    default: false

storm.lib.axon.wget
    Controls the ability to retrieve a file from URL and store it in the Axon.
    gate: cortex
    default: false

storm.lib.axon.wput
    Controls the ability to push a file from the Axon to a URL.
    gate: cortex
    default: false

storm.lib.cortex.httpapi.add
    Controls the ability to add a new Extended HTTP API on the Cortex.
    gate: cortex
    default: false

storm.lib.cortex.httpapi.del
    Controls the ability to delete an Extended HTTP API on the Cortex.
    gate: cortex
    default: false

storm.lib.cortex.httpapi.get
    Controls the ability to get or list Extended HTTP APIs on the Cortex.
    gate: cortex
    default: false
```

(continues on next page)

(continued from previous page)

```
storm.lib.cortex.httpapi.set
    Controls the ability to modify an Extended HTTP API on the Cortex.
    gate: cortex
    default: false

storm.lib.inet.http.proxy
    Permits a user to specify the proxy used with `$lib.inet.http` APIs.
    gate: cortex
    default: false

storm.lib.log.debug
    Controls the ability to log a debug level message.
    gate: cortex
    default: false

storm.lib.log.error
    Controls the ability to log a error level message.
    gate: cortex
    default: false

storm.lib.log.info
    Controls the ability to log a info level message.
    gate: cortex
    default: false

storm.lib.log.warning
    Controls the ability to log a warning level message.
    gate: cortex
    default: false

storm.lib.telepath.open
    Controls the ability to open an arbitrary telepath URL. USE WITH CAUTION.
    gate: cortex
    default: false

storm.lib.telepath.open.<scheme>
    Controls the ability to open a telepath URL with a specific URI scheme. USE WITH␣
↪CAUTION.
    gate: cortex
    default: false

storm.macro.add
    Controls access to add a storm macro.
    gate: cortex
    default: true

storm.macro.admin
    Controls access to edit/set/delete a storm macro.
    gate: cortex
    default: false

storm.macro.edit
```

(continues on next page)

(continued from previous page)

```
    Controls access to edit a storm macro.
    gate: cortex
    default: false

trigger.add
    Controls adding triggers.
    gate: cortex
    default: false

trigger.del
    Controls deleting triggers.
    gate: view
    default: false

trigger.get
    Controls listing/retrieving triggers.
    gate: trigger
    default: false

trigger.set
    Controls enabling, disabling, and modifying the query of a trigger.
    gate: view
    default: false

trigger.set.<property>
    Controls modifying specific trigger properties.
    gate: view
    default: false

trigger.set.doc
    Controls modifying the doc property of triggers.
    gate: trigger
    default: false

trigger.set.name
    Controls modifying the name property of triggers.
    gate: trigger
    default: false

trigger.set.user
    Controls modifying the user property of triggers.
    gate: cortex
    default: false

view
    Controls all view permissions.
    gate: cortex
    default: false

view.add
    Controls access to add a new view including forks.
    gate: cortex
```

(continues on next page)

(continued from previous page)

```
    default: false

view.del
    Controls access to delete a view.
    gate: view
    default: false

view.fork
    Controls access to fork a view.
    gate: view
    default: true

view.read
    Controls read access to view.
    gate: view
    default: false

view.set.<setting>
    Controls access to change view settings.
    gate: view
    default: false
    example: view.set.name
```

4.3.6 Optic Permissions

Commercial Synapse customers with the Optic UI may need to explicitly grant users or roles permission to some UI tools (such as Spotlight).

- See the Optic Deployment Guide for information on Optic deployment.
- See the Optic DevOps Guide for information on Optic permissions and other features.

Tip: You do not need to explicitly grant permissions to Optic itself. If you are creating and managing Synapse ("Cortex") users and roles via Optic, they have permission to access Optic by default.

4.3.7 Power-Up Permissions

Synapse **Power-Ups** have their own sets of permissions that must be granted to users or roles to allow them to use the Power-Up and any associated Storm commands. Specific permissions are documented in the **Admin Guide** section of the Power-Up documentation for the individual Power-Up.

Tip: While most Vertex-provided Power-Ups are part of the commercial Synapse offering, the following Rapid Power-Ups are also available for use with the community (open source) version of Synapse:

- Synapse-MISP
- Synapse-MITRE-ATT&CK
- Synapse-PSL (FQDN public suffix list)
- Synapse-TOR

4.3.8 Storm Runtime Permissions

When a user runs a Storm query **interactively** (e.g., in the Storm CLI or via the Optic Query Bar), or performs an action in the Optic UI (such as accessing a menu option), the query or action executes **with the permissions of the user,** based on the applicable user and role permissions and the current scope for the query or action.

There are a few cases of Storm runtime execution where different permissions are used that may require additional considerations.

Automation

Synapse includes the ability to automate Storm-based tasks using triggers, cron jobs, and / or macros. These elements are all impacted by permissions in various ways, including:

- who can create or manage automation (e.g., by default any user can create a macro, but explicit permissions are required to create triggers or cron jobs);
- who a given piece of automation runs as (e.g., macros run as the user who executes them, but triggers and cron jobs run as the user who created them).

Refer to the *Storm Reference - Automation* section of the *Synapse User Guide* for a detailed discussion of automation in Synapse (including permissions considerations).

Power-Ups

Power-Ups implement Storm packages and Storm services to provide additional functionality to Synapse. Power-Ups may be provided by The Vertex Project (as free or commercial offerings). Organizations may also develop their own custom Power-Ups.

Power-Ups commonly install Storm commands to allow users to make use of the additional capabilities of the Power-Up. In some cases, Power-Ups may need to access sensitive data (such as API keys or similar credentials) or perform actions (e.g., in adding nodes or applying tags) that some users would not be allowed to perform on their own.

Power-Ups can use privilege separation ("privsep") so that a limited subset of Power-Up capabilities can run with elevated privileges if necessary, with the remainder of the code running as the user who calls the Power-Up.

See the *Rapid Power-Up Development* section of the *Synapse Developer Guide* for additional details.

Note: Synapse Admins are typically only responsible for ensuring that the appropriate users and roles can use or run individual Power-Ups (see *Power-Up Permissions*). While Synapse Admins should be aware of privilege separation within a Power-Up as a best practice, implementation of privilege separation is left to Power-Up developers.

4.4 Add Extended Model Elements

The Synapse data model in a Cortex can be extended with custom forms or properties by using the model extension Storm Library (*$lib.model.ext*). Extended model forms and properties must have names beginning with an underscore (_) to avoid potential naming conflicts with built-in model elements.

Note: Extended model elements that are in-use (have nodes using the extended forms or properties) cannot be removed until all instances of that extended model element are removed. In other words, before removing extended forms any nodes created with that extended form must be delete first, and before removing extended properties any nodes with that extended property must have the property value removed.

4.4.1 Extended Forms

When adding a form, `$lib.model.ext.addForm` takes the following arguments:

formname
Name of the form, must begin with an underscore (_) and contain at least one colon (:).

basetype
The Synapse data model type for the form.

typeopts
A dictionary of type specific options.

typeinfo
A dictionary of info values for the form.

To add a new form named `_foocorp:name`, which contains string values which will be normalized to lowercase, with whitespace stripped from the beginning/end:

```
$typeopts = ({'lower': $lib.true, 'strip': $lib.true})
$typeinfo = ({'doc': 'Foocorp name.'})

$lib.model.ext.addForm(_foocorp:name, str, $typeopts, $typeinfo)
```

If the form is no longer in use and there are no nodes of this form in the Cortex, it can be removed with:

```
$lib.model.ext.delForm(_foocorp:name)
```

4.4.2 Extended Properties

When adding properties, `$lib.model.ext.addFormProp` takes the following arguments:

formname
Name of the form to add the property to, may be a built-in or extended model form.

propname
Relative name of the property, must begin with an underscore (_).

typedef
A tuple of (`type`, `typeopts`) which defines the type for the property

propinfo
A dictionary of info values for the property.

To add a property named `_score` to the `_foocorp:name` form which contains int values between 0 and 100:

```
$typeopts = ({'min': 0, 'max': 100})
$propinfo = ({'doc': 'Score for this name.'})

$lib.model.ext.addFormProp(_foocorp:name, _score, (int, $typeopts), $propinfo)
```

To add a property named `_aliases` to the `_foocorp:name` form which contains a unique array of `ou:name` values:

```
$typeopts = ({'type': 'ou:name', 'uniq': $lib.true})
$propinfo = ({'doc': 'Aliases for this name.'})

$lib.model.ext.addFormProp(_foocorp:name, _aliases, (array, $typeopts), $propinfo)
```

Properties may also be added to existing forms, for example, to add a property named `_classification` to `inet:fqdn` which must contain a string from a predefined set of values:

```
$typeopts = ({'enums': 'unknown,benign,malicious'})
$propinfo = ({'doc': 'Classification for this FQDN.'})

$lib.model.ext.addFormProp(inet:fqdn, _classification, (str, $typeopts), $propinfo)
```

4.4.3 Extended Universal Properties

Similar to `$lib.model.ext.addFormProp`, `$lib.model.ext.addUnivProp` takes the same `propname`, `typedef`, and `propinfo` arguments, but applies to all forms.

4.5 Manage Model Deprecations

As the Synapse Data Model grows and evolves, model elements (types, forms, and properties) may be deprecated and should no longer be used for new data modeling. The Storm `model.deprecated` commands can be used to prepare for the eventual removal of deprecated model elements.

4.5.1 Lock Deprecated Model Elements

The *model.deprecated.lock* command edits the lock status of deprecated model elements. Locked model elements can still be viewed or deleted, but can no longer be added. Attempting to add a locked model element will cause an `IsDeprLocked` error. The *model.deprecated.locks* command can be used to show the current lock status of all deprecated model elements.

Examples:

Lock the `ps:person:img` property:

```
storm> model.deprecated.lock ps:person:img
Locking: ps:person:img
```

Unlock the `ps:person:img` property:

```
storm> model.deprecated.lock --unlock ps:person:img
Unlocking: ps:person:img
```

Lock all deprecated model elements:

```
storm> model.deprecated.lock *
Locking all deprecated model elements.
```

4.5.2 Check for Deprecated Model Elements

The *model.deprecated.check* command checks for lock status and the existence of deprecated model elements in the Cortex. Warnings will be produced for any deprecated model elements which are unlocked or still in use in the Cortex. Once all warnings have been resolved, your Cortex will be ready for future model updates.

4.6 Configure a Mirrored Layer

Note: Mirrored layers are deprecated in 2.x and will be removed in 3.0.0. The layer.pull.add and layer.push.add commands can be used to configure streaming edits to/from layers in a separate Cortex.

Once a mirrored layer is configured, it will need to stream down the entire history of events from the upstream layer. During this process, the layer will be readable but writes will hang due to needing to await the write-back A Cortex may be configured to mirror a layer from a remote Cortex which will synchronize all edits from the remote layer and use write-back support to facilitate edits originating from the downstream layer. The mirrored layer will be an exact copy of the layer on the remote system including all edit history and will only allow changes which are first sent to the upstream layer.

When configuring a mirrored layer, you may choose to mirror from a remote layer *or* from the top layer of a remote view. If you choose to mirror from the top layer of a remote view, that view will have the opportunity to fire triggers and enforce model constraints on the changes being provided by the mirrored layer.

To specify a remote layer as the upstream, use a Telepath URL which includes the shared object `*/layer/<layeriden>` such as:

```
aha://cortex.loop.vertex.link/*/layer/8ea600d1732f2c4ef593120b3226dea3
```

To specify a remote view, use the shared object `*/view/<viewiden>` such as:

```
aha://cortex.loop.vertex.link/*/view/8ea600d1732f2c4ef593120b3226dea3
```

When you specify a `--mirror` option to the `layer.add` command or within a layer definition provided to the `$lib.layer.add()` Storm API the telepath URL will not be checked. This allows configuration of a remote layer or view which is not yet provisioned or is currently offline.

Note: To allow write access, the telepath URL must allow admin access to the remote Cortex due to being able to fabricate edit origins. The telepath URL may use aliased names or TLS client side certs to prevent credential disclosure.

Once a mirrored layer is configured, it will need to stream down the entire history of events from the upstream layer. During this process, the layer will be readable but writes will hang due to needing to await the write-back to be fully caught up to guarantee that edits are immediately observable like a normal layer. During that process, you may track progress by calling the `getMirrorStatus()` API on the `layer` object within the Storm runtime.

CHAPTER

FIVE

SYNAPSE DEPLOYMENT GUIDE

5.1 Introduction

This step-by-step guide will walk you through a production-ready Synapse deployment. Services will be configured to register with `AHA` for service discovery and to prepare for future devops tasks such as promoting a mirror to leader and provisioning future Synapse Advanced Power-Ups.

This guide will also walk you through deploying all Synapse services using TLS to authenticate both servers and clients using client-certificates to minimize the need for secrets management by eliminating passwords from all telepath URLs.

For the purposes of this guide, we will use `docker-compose` as a light-weight orchestration mechanism. The steps, configurations, and volume mapping guidance given in this guide apply equally to other container orchestration mechanisms such as Kubernetes but for simplicity's sake, this guide will only cover `docker-compose` based deployments.

Note: Due to known networking limitations of docker on Mac we do **not** support or recommend the use of Docker for Mac for testing or deploying production Synapse instances. Containers run within separate `docker-compose` commands will not be able to reliably communicate with each other.

Synapse services **require persistent storage**. Each `docker` container expects persistent storage to be available within the directory `/vertex/storage` which should be a persistent mapped volume. Only one container may run from a given volume at a time.

Note: To allow hosts to be provisioned on one system, this guide instructs you to disable HTTP API listening ports on all services other than the main Cortex. You may remove those configuration options if you are running on separate hosts or select alternate ports which do not conflict.

5.2 Prepare your Hosts

Ensure that you have an updated install of docker and docker-compose.

In order to help you run the Synapse service containers as a non-root user, Synapse service docker containers have been preconfigured with a user named `synuser` with UID `999`. You may replace `999` in the configs below, but keep in mind that doing so will result in the container not having a name for the user. We recommend that you do **not** use the Linux user `nobody` for this purpose.

Default kernel parameters on most Linux distributions are not optimized for database performance. We recommend adding the following lines to `/etc/sysctl.conf` on all systems being used to host Synapse services:

```
vm.swappiness=10
vm.dirty_expire_centisecs=20
vm.dirty_writeback_centisecs=20
```

See *Performance Tuning* for a list of additional tuning options.

We will use the directory `/srv/syn/` on the host systems as the base directory used to deploy the Synapse services. Each service will be deployed in separate `/srv/syn/<svcname>` directories. This directory can be changed to whatever you would like, and the services may be deployed to any host provided that the hosts can directly connect to each other. It is critical to performance that these storage volumes be low-latency. More latent storage mechanisms such as spinning disks, NFS, or EFS should be avoided!

We highly recommend that hosts used to run Synapse services deploy a log aggregation agent to make it easier to view the logs from the various containers in a single place.

When using AHA, you may run any of the **other** services on additional hosts as long as they can connect directly to the AHA service. You may also shutdown a service, move it's volume to a different host, and start it backup without changing anything.

5.3 Decide on a Name

Throughout the examples, we will be using `<yournetwork>` as the AHA network name which is also used as the common-name (CN) for the CA certificate. This should be changed to an appropriate network name used by your synapse deployment such as `syn.acmecorp.com`. We will use `<yournetwork>` in the following configs to specify locations which should be replaced with your selected AHA network name. For a **test** deployment which runs **all** docker containers on one host, you may use `loop.vertex.link`.

Note: It is important that you choose a name and stick with it for a given deployment. Once we begin generating host and service account certificates, changing this name will be difficult.

5.4 Deploy AHA Service

The AHA service is used for service discovery and acts as a CA to issue host/user certificates used to link Synapse services. Other Synapse services will need to be able to resolve the IP address of the AHA service by name, so it is likely that you need to create a DNS A/AAAA record in your existing resolver. When you are using AHA, the only host that needs DNS or other external name resolution is the AHA service.

Note: It is important to ensure that `aha.<yournetwork>` is resolvable via DNS or docker container service name resolution from within the container environment! There are configuration options you may use if this is impossible, but the configuration is far simpler if we can make this assumption.

Create the container directory:

```
mkdir -p /srv/syn/aha/storage
```

Create the `/srv/syn/aha/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  aha:
    user: "999"
    image: vertexproject/synapse-aha:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        - SYN_AHA_HTTPS_PORT=null
        - SYN_AHA_AHA_NAME=aha
        - SYN_AHA_AHA_NETWORK=<yournetwork>
        - SYN_AHA_DMON_LISTEN=ssl://aha.<yournetwork>?ca=<yournetwork>
        - SYN_AHA_PROVISION_LISTEN=ssl://aha.<yournetwork>:27272
```

Note: Don't forget to replace `<yournetwork>` with your chosen network name!

Change ownership of the storage directory to the user you will use to run the container:

```
chown -R 999 /srv/syn/aha/storage
```

Start the container using `docker-compose`:

```
docker-compose -f /srv/syn/aha/docker-compose.yaml pull
docker-compose -f /srv/syn/aha/docker-compose.yaml up -d
```

To view the container logs at any time you may run the following command on the *host* from the `/srv/syn/aha` directory:

```
docker-compose logs -f
```

You may also execute a shell inside the container using `docker-compose` from the `/srv/syn/aha` directory on the *host*. This will be necessary for some of the additional provisioning steps:

```
docker-compose exec aha /bin/bash
```

5.5 Deploy Axon Service

In the Synapse service architecture, an Axon provides a place to store arbitrary bytes/files as binary blobs and exposes APIs for streaming files in and out regardless of their size. Given sufficient file system size, an Axon can be used to efficiently store and retrieve very large files as well as a high number (easily billions) of files.

Inside the AHA container

Generate a one-time use provisioning URL:

```
python -m synapse.tools.aha.provision.service 00.axon
```

These one-time use URLs are used to connect to the Aha service, retrieve configuration data, and provision SSL certificates for the service. When this is done, the service records that the URL has been used in its persistent storage, and will not attempt to perform the provisioning process again unless the URL changes. If the provisioning URL is reused, services will encounter **NoSuchName** errors and fail to start up - this indicates a service has attempted to re-use the one-time use URL!

Note: We strongly encourage you to use a numbered hierarchical naming convention for services where the first part of the name is a 0 padded number and the second part is the service type. The above example `00.axon` will allow you to deploy mirror instances in the future, such as `01.axon`, where the AHA name `axon.<yournetwork>` will automatically resolve to which ever one is the current leader.

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container directory:

```
mkdir -p /srv/syn/00.axon/storage
chown -R 999 /srv/syn/00.axon/storage
```

Create the `/srv/syn/00.axon/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  00.axon:
    user: "999"
    image: vertexproject/synapse-axon:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        # disable HTTPS API for now to prevent port collisions
        - SYN_AXON_HTTPS_PORT=null
        - SYN_AXON_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Note: Don’t forget to replace your one-time use provisioning URL!

Start the container:

```
docker-compose --file /srv/syn/00.axon/docker-compose.yaml pull
docker-compose --file /srv/syn/00.axon/docker-compose.yaml up -d
```

5.6 Deploy JSONStor Service

Inside the AHA container

Generate a one-time use provisioning URL:

```
python -m synapse.tools.aha.provision.service 00.jsonstor
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container directory:

```
mkdir -p /srv/syn/00.jsonstor/storage
chown -R 999 /srv/syn/00.jsonstor/storage
```

Create the /srv/syn/00.jsonstor/docker-compose.yaml file with contents:

```
version: "3.3"
services:
  00.jsonstor:
    user: "999"
    image: vertexproject/synapse-jsonstor:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        # disable HTTPS API for now to prevent port collisions
        - SYN_JSONSTOR_HTTPS_PORT=null
        - SYN_JSONSTOR_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=
↪<sha256>
```

Note: Don't forget to replace your one-time use provisioning URL!

Start the container:

```
docker-compose --file /srv/syn/00.jsonstor/docker-compose.yaml pull
docker-compose --file /srv/syn/00.jsonstor/docker-compose.yaml up -d
```

5.7 Deploy Cortex Service

Inside the AHA container

Generate a one-time use provisioning URL:

```
python -m synapse.tools.aha.provision.service 00.cortex
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container directory:

```
mkdir -p /srv/syn/00.cortex/storage
chown -R 999 /srv/syn/00.cortex/storage
```

Create the `/srv/syn/00.cortex/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  00.cortex:
    user: "999"
    image: vertexproject/synapse-cortex:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        - SYN_CORTEX_AXON=aha://axon...
        - SYN_CORTEX_JSONSTOR=aha://jsonstor...
        - SYN_CORTEX_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Note: Don't forget to replace your one-time use provisioning URL!

Note: The values `aha://axon...` and `aha://jsonstor...` can be used as-is without changing them because the AHA network (provided by the provisioning server) is automatically subtituted in any `aha://` scheme URL ending with `...`

Start the container:

```
docker-compose --file /srv/syn/00.cortex/docker-compose.yaml pull
docker-compose --file /srv/syn/00.cortex/docker-compose.yaml up -d
```

Remember, you can view the container logs in real-time using:

```
docker-compose --file /srv/syn/00.cortex/docker-compose.yaml logs -f
```

5.8 Deploy Cortex Mirror (optional)

Inside the AHA container

Generate a one-time use URL for provisioning from *inside the AHA container*:

```
python -m synapse.tools.aha.provision.service 01.cortex --mirror cortex
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container storage directory:

```
mkdir -p /srv/syn/01.cortex/storage
chown -R 999 /srv/syn/01.cortex/storage
```

Create the `/srv/syn/01.cortex/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  01.cortex:
    user: "999"
    image: vertexproject/synapse-cortex:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        - SYN_CORTEX_AXON=aha://axon...
        - SYN_CORTEX_JSONSTOR=aha://jsonstor...
        # disable HTTPS API for now to prevent port collisions
        - SYN_CORTEX_HTTPS_PORT=null
        - SYN_CORTEX_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Note: Don't forget to replace your one-time use provisioning URL!

Start the container:

```
docker-compose --file /srv/syn/01.cortex/docker-compose.yaml pull
docker-compose --file /srv/syn/01.cortex/docker-compose.yaml up -d
```

Note: If you are deploying a mirror from an existing large Cortex, this startup may take a while to complete initialization.

5.9 Enroll CLI Users

A Synapse user is generally synonymous with a user account on the Cortex. To bootstrap CLI users who will have Cortex access using the Telepath API, we will need to add them to the Cortex and generate user certificates for them. To add a new admin user to the Cortex, run the following command from **inside the Cortex container**:

```
python -m synapse.tools.moduser --add --admin true visi
```

Note: If you are a Synapse Enterprise customer, using the Synapse UI with SSO, the admin may now login to the Synapse UI. You may skip the following steps if the admin will not be using CLI tools to access the Cortex.

Then we will need to generate a one-time use URL they may use to generate a user certificate. Run the following command from **inside the AHA container** to generate a one-time use URL for the user:

```
python -m synapse.tools.aha.provision.user visi
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Then the **user** may run:

```
python -m synapse.tools.aha.enroll ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Once they are enrolled, they will have a user certificate located in `~/.syn/certs/users` and their telepath configuration located in `~/.syn/telepath.yaml` will be updated to reflect the use of the AHA server. From there the user should be able to use standard Synapse CLI tools using the `aha://` URL such as:

```
python -m synapse.tools.storm aha://visi@cortex.<yournetwork>
```

5.10 Configure a Storm Query Pool (optional)

A Cortex may be configured to use a pool of mirrors in order to offload Storm query execution and distribute query load among a configurable group of mirrors. We will assume you have configured two additional mirrors named `01.cortex...` and `02.cortex...` using the process described in the previous *Deploy Cortex Mirror (optional)* step. In our example, we will also assume that the mirrors will be used for both query parallelism and for graceful promotions to minimize downtime during upgrades and optimization.

The following commands are run using the Storm CLI tool discussed in the *Enroll CLI Users* section. First, use the Storm CLI to run the `aha.pool.add` command to create a new AHA pool:

```
aha.pool.add pool00.cortex...
```

Then add the Cortex leader as well as the two mirrors to the pool:

```
aha.pool.svc.add pool00.cortex... 00.cortex...
aha.pool.svc.add pool00.cortex... 01.cortex...
aha.pool.svc.add pool00.cortex... 02.cortex...
```

Then configure the Cortex to use the newly created AHA service pool:

```
cortex.storm.pool.set aha://pool00.cortex...
```

Now your Cortex will distribute Storm queries across the available mirrors. You may add or remove mirrors from the pool at any time using the `aha.pool.svc.add` and `aha.pool.svc.del` commands and the pool topology updates will be automatically sent. You may want to review some of the command options to adjust timeouts for your environment.

If you wish to remove the pool configuration from the Cortex you may use the `cortex.storm.pool.del` command.

5.11 What's next?

See the *Synapse Admin Guide* for instructions on performing application administrator tasks. See the *Synapse Devops Guide* for instructions on performing various maintenance tasks on your deployment!

CHAPTER

SIX

SYNAPSE DEVOPS GUIDE

6.1 Overview

6.1.1 Docker Images

Each Synapse service is distributed as a `docker` image which contains all the dependencies required to run the service. For the open-source Synapse images, the tag `:v2.x.x` will always be present on the most recent supported release. Image names are specified in each service specific section below.

Synapse services **require persistent storage**. Each `docker` container expects persistent storage to be available within the directory `/vertex/storage` which should be a persistent mapped volume. Only one container may run from a given volume at a time.

6.1.2 cell.yaml

Each Synapse service has one configuration file, `cell.yaml`, which is located in the service storage directory, typically `/vertex/storage/cell.yaml` in the `docker` images. Configuration options are specified in YAML format using the same syntax as their documentation, for example:

```
aha:name: cortex
aha:network: loop.vertex.local
```

6.1.3 Environment Variables

Synapse services may also be configured using environment variables specified in their documentation. The value will be parsed as a YAML value to allow structured data to be specified via environment variables and then subject to normal configuration schema validation.

6.1.4 HTTPS Certificates

Synapse services that expose HTTPS APIs will automatically generate a self-signed certificate and key if they are not found at `sslcert.pem` and `sslkey.pem` in the service storage directory. At any time, you can replace these self-signed files with a certificate and key generated using *easycert* or generated and signed by an external CA.

6.2 Common Devops Tasks

6.2.1 Generating a Backup

Note: If you are a Synapse Enterprise customer you should deploy the Synapse-Backup Advanced Power-Up.

It is strongly recommended that users schedule regular backups of all services deployed within their Synapse ecosystem. Each service must be backed up using either the **live** backup tool `synapse.tools.livebackup` or the offline backup tool `synapse.tools.backup`.

For a production deployment similar to the one described in the *Synapse Deployment Guide* you can easily run the backup tool by executing a shell **inside** the docker container. For example, if we were generating a backup of the Cortex we would:

```
cd /srv/syn/00.cortex
docker-compose exec 00.cortex /bin/bash
```

And from the shell executed within the container:

```
python -m synapse.tools.livebackup
```

This will generate a backup in a time stamp directory similar to:

```
/vertex/storage/backups/20220422094622
```

Once the backup directory is generated you may exit the docker shell and the backup will be accessible from the **host** file system as:

```
/srv/syn/00.cortex/storage/backups/20220422094622
```

At this point it is safe to use standard tools like `mv`, `tar`, and `scp` on the backup folder:

```
mv /srv/syn/00.cortex/storage/backups/20220422094622 /nfs/backups/00.cortex/
```

Note: It is important that you use `synapse.tools.livebackup` to ensure a transactionally consistant backup.

Note: When taking a backup of a service, the backup is written by the service locally to disk. This may take up storage space equal to the current size of the service. If the service does not have the `backup:dir` option configured for a dedicated backup directory (or volume), this backup is made to `/vertex/storage/backups` by default. If the volume backing `/vertex/storage` reaches a maximum capacity, the backup process will fail.

To avoid this from being an issue, when using the default configuration, make sure services do not exceed 50% of their storage utilization. For example, a Cortex that has a size of 32GB of utilized space may take up 32GB during a backup. The volume backing `/vertex/storage` should be at least 64GB in size to avoid issues taking backups.

It is also worth noting that the newly created backup is a defragmented / optimized copy of the databases. We recommend occasionally scheduling a maintenance window to create a "cold backup" using the offline `synapse.tools.backup` command with the service offline and deploy the backup copy when bringing the service back online. Regularly performing this "restore from cold backup" procedure can dramatically improve performance and resource utilization.

6.2.2 Restoring a Backup

In the hopefully unlikely event that you need to restore a **Synapse** service from a backup the process is fairly simple. For a production deployment similar to the one described in *Synapse Deployment Guide* and assuming we moved the backup file as described in *Generating a Backup*:

```
cd /srv/syn/00.cortex
docker-compose down
mv storage storage.broken
cp -R /nfs/backups/00.cortex/20220422094622 storage
docker-compose up -d
```

Then you can tail the logs to ensure the service is fully restored:

```
cd /srv/syn/00.cortex
docker-compose logs -f
```

6.2.3 Promoting a Mirror

Note: To gracefully promote a mirror to being the leader, your deployment must include AHA based service discovery as well as use TLS client-certificates for service authentication.

To gracefully promote a mirror which was deployed in a similar fashion to the one described in *Synapse Deployment Guide* you can use the built-in promote tool `synapse.tools.promote`. Begin by executing a shell within the mirror container:

```
cd /srv/syn/01.cortex
docker-compose exec 01.cortex /bin/bash
```

And from the shell executed within the container:

```
python -m synapse.tools.promote
```

Once completed, the previous leader will now be configured as a follower of the newly promoted leader.

Note: If you are promoting the follower due to a catastrophic failure of the previous leader, you may use the command `synapse.tools.promote --failure` to force promotion despite not being able to carry out a graceful handoff. It is **critical that you not bring the previous leader back online** once this has been done. To regain redundancy, deploy a new mirror using the AHA provisioning process described in the *Synapse Deployment Guide*.

6.2.4 Updating Services

Updating a Synapse service requires pulling the newest docker image and restarting the container. For Synapse services which have mirrors deployed, you must ensure that the mirrors are updated first so that any newly introduced change messages can be consumed. If you are using a mirrors-of-mirrors tree topology, the update should be deployed in a "leafs first" order.

Continuing with our previous example from the *Synapse Deployment Guide* we would update the mirror `01.cortex` first:

```
cd /srv/syn/01.cortex
docker-compose pull
docker-compose down
docker-compose up -d
```

After ensuring that the mirror has come back online and is fully operational, we will update the leader which may include a *Data Migration* while it comes back online:

```
cd /srv/syn/00.cortex
docker-compose pull
docker-compose down
docker-compose up -d
```

Note: Once a Synapse service update has been deployed, you may **NOT** revert to a previous version!

Data Migration

When a Synapse release contains a data migration for a part of the Synapse data model, the Changelog will indicate what component is being migrated and why. This will be made under the `Automated Migrations` header, at the top of the changelog.

Automatic data migrations may cause additional startup times on the first boot of the version. When beginning a data migration, a WARNING level log message will be printed for each stage of the migration:

```
beginning model migration -> (0, 2, 8)
```

Once complete, a WARNING level log message will be issued:

```
...model migrations complete!
```

Note: Please ensure you have a tested backup available before applying these updates.

Model Flag Day

Periodically, a Synapse release will include small, but technically backward incompatible, changes to the data model. All such migrations will include a `Model Flag Day` heading in the Changelog with a detailed description of each change to the data model. Additionally, the release will execute an in-place migration to modify data to confirm with model updates. If necessary, any data that can not be migrated automatically will be saved to a location documented within the detailed description.

When we release a Synapse version containing a `Model Flag Day` update, we will simultaneously release updates to any effected Power-Ups.

Examples of potential `Model Flag Day` changes:

- Removing a previously deprecated property
- Specifying a more specific type for a property to allow pivoting
- Tightening type normalization constraints of a property

It is **highly** recommended that production deployments have a process for testing custom storm code in a staging environment to help identify any tweaks that may be necessary due to the updated data model.

Note: Please ensure you have a tested backup available before applying these updates.

6.2.5 Configure Logging

Synapse services support controlling log verbosity via the `SYN_LOG_LEVEL` environment variable. The following values may be used: `CRITCAL`, `ERROR`, `WARNING`, `INFO`, and `DEBUG`. For example:

```
SYN_LOG_LEVEL=INFO
```

To enable JSON structured logging output suitable for ingest and indexing, specify the following environment variable to the `docker` container:

```
SYN_LOG_STRUCT=true
```

These structured logs are designed to be easy to ingest into third party log collection platforms. They contain the log message, level, time, and metadata about where the log message came from:

```
{
  "message": "log level set to INFO",
  "logger": {
    "name": "synapse.lib.cell",
    "process": "MainProcess",
    "filename": "common.py",
    "func": "setlogging"
  },
  "level": "INFO",
  "time": "2021-06-28 15:47:54,825"
}
```

When exceptions are logged with structured logging, we capture additional information about the exception, including the entire traceback. In the event that the error is a Synapse Err class, we also capture additional metadata which was attached to the error. In the following example, we also have the query text, username and user iden available in the log message pretty-printed log message:

```
{
  "message": "Error during storm execution for { || }",
  "logger": {
    "name": "synapse.lib.view",
    "process": "MainProcess",
    "filename": "view.py",
    "func": "runStorm"
  },
  "level": "ERROR",
  "time": "2021-06-28 15:49:34,401",
  "err": {
    "efile": "coro.py",
    "eline": 233,
    "esrc": "return await asyncio.get_running_loop().run_in_executor(forkpool, _runtodo,␣
↪todo)",
```

(continues on next page)

(continued from previous page)

```
    "ename": "forked",
    "at": 1,
    "text": "||",
    "mesg": "No terminal defined for '|' at line 1 col 2.  Expecting one of: #, $, (, *,␣
↪+ or -, -(, -+>, -->, ->, :, <(, <+-, <-, <--, [, break, command name, continue, fini,␣
↪for, function, if, init, property name, return, switch, while, whitespace or comment,␣
↪yield, {",
    "etb": ".... long traceback ...",
    "errname": "BadSyntax"
  },
  "text": "||",
  "username": "root",
  "user": "3189065f95d3ab0a6904e604260c0be2"
}
```

Custom date formatting strings can also be provided by setting the SYN_LOG_DATEFORMAT string. This is expected to be a strftime format string. The following shows an example of setting this value:

```
SYN_LOG_DATEFORMAT="%d%m%Y %H:%M:%S"
```

produces the following output:

```
28062021 15:48:01 [INFO] log level set to DEBUG [common.
↪py:setlogging:MainThread:MainProcess]
```

This will also be used to format the `time` key used for structured logging.

> **Warning:** Milliseconds are not available when using the date formatting option. This will result in a loss of precision for the timestamps that appear in log output.

6.2.6 Configure Free Space Requirement

To avoid the risk of data corruption due to lack of disk space, Synapse services periodically check the amount of free space available and will switch to read-only mode if they are below a minimum threshold. This threshold can be controlled via the `limit:disk:free` configuration option, and is set to 5% free space by default.

If the available free space goes below the minimum threshold, the service will continue the free space checks and re-enable writes if the available space returns above the threshold.

6.2.7 Performance Tuning

Performance tuning Synapse services is very similar to performance tuning other database systems like PostgreSQL or MySQL. Recommendations for good performance for other database systems may also apply to Synapse services. Database systems run best when given as much RAM as possible. Under **ideal** circumstances, the amount of RAM exceeds the total database storage size.

Minimizing storage latency is important for a high performance Synapse service. Locating the storage volume backed by a mechanical hard drive is **strongly** discouraged. For the same reason, running Synapse services from an NFS file system (including NFS-based systems like AWS EFS) is **strongly** discouraged.

The default settings of most Linux-based operating systems are not set for ideal performance.

Consider setting the following Linux system variables. These can be set via /etc/sysctl.conf, the sysctl utility, or writing to the /proc/sys file system.

vm.swappiness=10
Reduce preference for kernel to swap out memory-mapped files.

vm.dirty_expire_centisecs=20
Define "old" data to be anything changed more than 200 ms ago.

vm.dirty_writeback_centisecs=20
Accelerate writing "old" data back to disk.

vm.dirty_background_ratio=2
This is expressed as a percentage of total RAM in the system. After the total amount of dirty memory exceeds this threshold, the kernel will begin writing it to disk in the background. We want this low to maximize storage I/O throughput utilization.

This value is appropriate for systems with 128 GiB RAM. For systems with less RAM, this number should be larger, for systems with more, this number may be smaller.

vm.dirty_ratio=4
This is expressed as a percentage of total RAM in the system. After the total amount of dirty memory exceeds this threshold, all writes will become synchronous, which means the Cortex will "pause" waiting for the write to complete. To avoid large sawtooth-like behavior, this value should be low.

This value is appropriate for systems with 128 GiB RAM. For systems with less RAM, this number should be larger, for systems with more, this number may be smaller.

This setting is particularly important for systems with lots of writing (e.g. making new nodes), lots of RAM, and relatively slow storage.

6.2.8 Managing Users and Roles

Adding Users

Managing users and service accounts in the Synapse ecosystem is most easily accomplished using the `moduser` tool executed from **within** the service `docker` container. In this example we add the user `visi` as an admin user to the Cortex by running the following command from **within the Cortex container**:

```
python -m synapse.tools.moduser --add --admin visi
```

If the deployment is using AHA and TLS client certificates and the user will be connecting via the Telepath API using the *storm* CLI tool, will also need to provision a user TLS certificate for them. This can be done using the `synapse.tools.aha.provision.user` command from **within the AHA container**:

```
python -m synapse.tools.aha.provision.user visi
```

Which will produce output similar to:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Note: The enrollment URL may only be used once. It should be given to the user using a secure messaging system to prevent an attacker from using it before the user.

Once the one-time enrollment URL has been passed along to the user, the **user must run an enrollment command** to configure their environment to use the AHA server and generate a user certificate from the host they will be using to run the Storm CLI:

```
python -m synapse.tools.aha.enroll ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Once they are enrolled, the user can connect using the Telepath URL `aha://cortex.<yournetwork>`:

```
python -m synapse.tools.storm aha://cortex.<yournetwork>
```

6.2.9 Updating to AHA and Telepath TLS

If you have an existing deployment which didn't initially include AHA and Telepath TLS, it can easily be deployed and configured after the fact. However, as services move to TLS it will **break existing telepath URLs** that may be in use, so you should test the deployment before updating your production instance.

To move to AHA, first deploy an AHA service as discussed in the *Synapse Deployment Guide*. For each service, you may then run the `provision` tool as described and add the `aha:provision` configuration option to the `cell.yaml` or use the service specific environment variable to prompt the service to provision itself.

Note: It is recommended that you name your services with leading numbers to prepare for an eventual mirror deployment.

For example, to add an existing Axon to your new AHA server, you would execute the following from **inside the AHA container**:

```
python -m synapse.tools.aha.provision 00.axon
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Then add the following entry to the Axon's `cell.conf`:

```
aha:provision: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Or add the following environment variable to your orchestration:

```
SYN_AXON_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

Then restart the Axon container. As it restarts, the service will generate user and host certificates and update it's `cell.yaml` file to include the necessary AHA configuration options. The `dmon:listen` option will be updated to reflect the use of SSL/TLS and the requirement to use client certificates for authentication. As additional services are provisioned, you may update the URLs they use to connect to the Axon to `aha://axon...`.

Note: When specifying a connection string using AHA, you can append a `mirror=true` parameter to the connection string (e.g. `aha://cortex...?mirror=true`) to cause AHA to prefer connecting to a service mirror rather than the leader (if mirrors are available).

6.2.10 Deployment Options

The following are some additional deployment options not covered in the *Synapse Deployment Guide*.

Note: These examples assume the reader has reviewed and understood the Synapse Deployment Guide.

Telepath Listening Port

If you need to deploy a service to have Telepath listen on a specific port, you can use the provision tool to specify the port to bind. This example will show deploying the Axon to a specific Telepath listening port.

Inside the AHA container

Generate a one-time use provisioning URL, with the `--dmon-port` option:

```
python -m synapse.tools.aha.provision.service --dmon-port 30001 01.axon
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container directory:

```
mkdir -p /srv/syn/01.axon/storage
chown -R 999 /srv/syn/01.axon/storage
```

Create the `/srv/syn/01.axon/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  01.axon:
    user: "999"
    image: vertexproject/synapse-axon:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        # disable HTTPS API for now to prevent port collisions
        - SYN_AXON_HTTPS_PORT=null
        - SYN_AXON_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

After starting the service, the Axon will now be configured to bind its Telepath listening port to 30001. This can be seen in the services `cell.yaml` file.

```
---
aha:name: 01.axon
aha:network: <yournetwork>
aha:provision: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
aha:registry:
- ssl://root@aha.<yournetwork>
```

(continues on next page)

(continued from previous page)

```
aha:user: root
dmon:listen: ssl://0.0.0.0:30001?hostname=01.axon.<yournetwork>&ca=
↪<yournetwork>
...
```

HTTPS Listening Port

If you need to deploy a service to have HTTPs listen on a specific port, you can use the provision tool to specify the port to bind. This example will show deploying the Cortex to a specific HTTPS listening port.

Inside the AHA container

Generate a one-time use provisioning URL, with the `--https-port` option:

```
python -m synapse.tools.aha.provision.service --https-port 8443 02.cortex
```

You should see output that looks similar to this:

```
one-time use URL: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

On the Host

Create the container directory:

```
mkdir -p /srv/syn/02.cortex/storage
chown -R 999 /srv/syn/02.cortex/storage
```

Create the `/srv/syn/01.axon/docker-compose.yaml` file with contents:

```
version: "3.3"
services:
  02.cortex:
    user: "999"
    image: vertexproject/synapse-axon:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        - SYN_CORTEX_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

After starting the service, the Cortex will now be configured to bind its HTTPS listening port to 8443. This can be seen in the services `cell.yaml` file.

```
---
aha:name: 02.cortex
aha:network: <yournetwork>
aha:provision: ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
aha:registry:
- ssl://root@aha.<yournetwork>
aha:user: root
dmon:listen: ssl://0.0.0.0:0?hostname=02.cortex.<yournetwork>&ca=<yournetwork>
https:port: 8443
...
```

6.2.11 Trimming the Nexus Log

The Nexus log can be trimmed to reduce the storage size of any Synapse Service that has Nexus logging enabled. This is commonly done before taking backups to reduce to their size.

For a Cortex **without** any mirrors, this is best accomplished in Storm via the following query:

```
$lib.cell.trimNexsLog()
```

The Storm API call will rotate the Nexus log and then delete the older entries.

If the Cortex is mirrored, a list of Telepath URLs of all mirrors must be provided. This ensures that all mirrors have rotated their Nexus logs before the cull operation is executed.

Warning: If this list is ommitted, or incorrect, the mirrors may become de-synchronized which will require a re-deployment from a backup of the upstream.

The Telepath URLs can be provided to the Storm API as follows:

```
$mirrors = ("aha://01.cortex...", "aha://02.cortex...")
$lib.cell.trimNexsLog(consumers=$mirrors)
```

6.2.12 Viewing Deprecation Warnings

When functionality in Synapse is deprecated, it is marked with the standard Python warnings mechanism to note that it is deprecated. Deprecated functionality is also noted in service changelogs as well. To view these warnings in your environment, you can set the PYTHONWARNINGS environment variable to display them. The following shows this being enabled for a Cortex deployment:

```
version: "3.3"
services:
  00.cortex:
    user: "999"
    image: vertexproject/synapse-cortex:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
    environment:
        - SYN_CORTEX_AXON=aha://axon...
        - SYN_CORTEX_JSONSTOR=aha://jsonstor...
        - PYTHONWARNINGS=default::DeprecationWarning:synapse.common
```

With this set, our deprecation warnings are emitted the first time the deprecated functionality is used. For example, if a remote caller uses the eval() API on a Cortex, it would log the following message:

```
/usr/local/lib/python3.8/dist-packages/synapse/common.py:913: DeprecationWarning:
↪"CoreApi.eval" is deprecated in 2.x and will be removed in 3.0.0
  warnings.warn(mesg, DeprecationWarning)
```

This would indicate the use of a deprecated API.

6.2.13 Entrypoint Hooking

Synapse service containers provide two ways that users can modify the container startup process, in order to execute their own scripts or commands.

The first way to modify the startup process is using a script that executes before services start. This can be configured by mapping in a file at `/vertex/boothooks/preboot.sh` and making sure it is marked as an executable. If this file is present, the script will be executed prior to booting the service. If this does not return `0`, the container will fail to start up.

One example for using this hook is to use `certbot` to create HTTPS certificates for a Synapse service. This example assumes the Cortex is running as root, so that certbot can bind port 80 to perform the `http-01` challenge. Non-root deployments may require additional port mapping for a given deployment.

Create a boothooks directory:

```
mkdir -p /srv/syn/00.cortex/bookhooks
```

Copy the following script to `/srv/syn/cortex/bookhooks/preboot.sh` and use `chmod` to mark it as an executable file:

```
#!/bin/bash

# Certbot preboot example
# Author: william.gibb@vertex.link

# This script is an example of using Let's Encrypt certbot tool to generate
# an HTTPS certificate for a Synapse service.
#
# This creates and stores a Python venv in the
# /vertex/storage/preboot/letsencrypt/venv directory, so the certbot
# tool is installed once in a separate python environment, and cached in
# a mapped volume.
#
# Once the venv is setup, certbot is used to create and potentially renew
# an HTTPS certificate. This certificate and private key are then copied to
# the locations in /vertex/storage where Synapse services assume they will
# find the HTTPS keys.
#
# certbot does use a random backoff timer when performing a renewal. There may
# be a random delay when starting a service when the certificate needs to be
# renewed.
#
# Required Environment variables:
#
# CERTBOT_HOSTNAME - the hostname that certbot will generate a certificate for.
# CERTBOT_EMAIL - the email address used with certbot.
#
# Optional Environment variables:
#
# CERTBOT_ARGS - additional args passed to the "certbot certonly" and
#   "certbot renew" commands.
#

# set -x # echo commands
```

(continues on next page)

(continued from previous page)

```
set -e # exit on nonzero

BASEDIR=/vertex/preboot
DSTKEY=/vertex/storage/sslkey.pem
DSTCRT=/vertex/storage/sslcert.pem

if [ -z ${CERTBOT_HOSTNAME} ]; then
    echo "CERTBOT_HOSTNAME env var is unset"
    exit 1
fi

if [ -z ${CERTBOT_EMAIL} ]; then
    echo "CERTBOT_EMAIL env var is unset"
    exit 1
fi

LEDIR=$BASEDIR/letsencrypt

CONFDIR=$LEDIR/conf
LOGSDIR=$LEDIR/logs
WORKDIR=$LEDIR/work
VENV=$LEDIR/venv

mkdir -p $LOGSDIR
mkdir -p $CONFDIR
mkdir -p $WORKDIR

CERTBOT_DIR_ARGS=" --work-dir ${WORKDIR} --logs-dir=${LOGSDIR} --config-dir=${CONFDIR} "

KEYFILE="${CONFDIR}/live/${CERTBOT_HOSTNAME}/privkey.pem"
CERTFILE="${CONFDIR}/live/${CERTBOT_HOSTNAME}/fullchain.pem"

# Create a python venv, activate it, and install certbot and supporting tools.
if [ ! -d $VENV ]; then
    echo "Creating venv and installing certbot"
    python3 -m venv --without-pip --copies $VENV

    . $VENV/bin/activate

    python3 -c "import urllib.request as ur; ur.urlretrieve('https://bootstrap.pypa.io/
↪get-pip.py', '/tmp/get-pip.py')"
    python3 /tmp/get-pip.py  # installs pip, wheel, setuptools
    python3 -m pip install --no-cache-dir "certbot==2.6.0"

else

    echo "Activating venv"
    . $VENV/bin/activate

fi

if [ ! -f ${KEYFILE} ]; then
```

(continues on next page)

(continued from previous page)

```
    certbot -n ${CERTBOT_DIR_ARGS} certonly --agree-tos --email ${CERTBOT_EMAIL} --
↪standalone -d ${CERTBOT_HOSTNAME} ${CERTBOT_ARGS:-}

    if [ $? -ne 0 ]; then
        echo "Error running certbot"
        exit 1
    fi

fi

certbot -n ${CERTBOT_DIR_ARGS} renew --standalone ${CERTBOT_ARGS:-}

if [ $? -ne 0 ]; then
    echo "Error checking certificate renewal"
    exit 1
fi

echo "Copying certificates"

cp ${KEYFILE} ${DSTKEY}
cp ${CERTFILE} ${DSTCRT}

echo "Done setting up HTTPS certificates"
```

That directory will be mounted at `/vertex/boothooks`. The following docker-compose file shows mounting that directory into the container and setting environment variables for the script to use:

```
version: "3.3"
services:
  00.cortex:
    image: vertexproject/synapse-cortex:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
        - ./storage:/vertex/storage
        - ./boothooks:/vertex/boothooks
    environment:
        SYN_LOG_LEVEL: "DEBUG"
        SYN_CORTEX_STORM_LOG: "true"
        SYN_CORTEX_AHA_PROVISION: "ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
↪"
        CERTBOT_HOSTNAME: "cortex.acme.corp"
        CERTBOT_EMAIL: "user@acme.corp"
```

When started, the container will attempt to run the script before starting the Cortex service.

The second way to modify a container startup process is running a script concurrently to the service. This can be set by mapping in a file at `/vertex/boothooks/concurrent.sh`, also as an executable file. If this file is present, the script is executed as a backgrounded task prior to starting up the Synapse service. This script would be stopped when the container is stopped.

Note: If a volume is mapped into `/vertex/boothooks/` it will not be included in any backups made by a Synapse service using the backup APIs. Making backups of any data persisted in these locations is the responsibility of the operator configuring the container.

6.2.14 Containers with Custom Users

By default, Synapse service containers will work running as `root` (uid 0) and `synuser` (uid 999) without any modification. In order to run a Synapse service container as a different user that is not built into the container by default, the user, group and home directory need to be added to the image. This can be done with a custom Dockerfile to modify a container. For example, the following Dockerfile would add the user `altuser` to the Container with a user id value of 8888:

```
FROM vertexproject/synapse-cortex:v2.x.x
RUN set -ex \
&& groupadd -g 8888 altuser \
&& useradd -r --home-dir=/home/altuser -u 8888 -g altuser --shell /bin/bash altuser \
&& mkdir -p /home/altuser \
&& chown 8888:8888 /home/altuser
```

Running this with a `docker build` command can be used to create the image `customcortex:v2.x.x`:

```
$ docker build -f Dockerfile --tag  customcortex:v2.x.x .
Sending build context to Docker daemon  4.608kB
Step 1/2 : FROM vertexproject/synapse-cortex:v2.113.0
---> 8a2dd3465700
Step 2/2 : RUN set -ex && groupadd -g 8888 altuser && useradd -r --home-dir=/home/
↪altuser -u 8888 -g altuser --shell /bin/bash altuser && mkdir -p /home/altuser &&␣
↪chown 8888:8888 /home/altuser
---> Running in 9c7b30365c2d
+ groupadd -g 8888 altuser
+ useradd -r --home-dir=/home/altuser -u 8888 -g altuser --shell /bin/bash altuser
+ mkdir -p /home/altuser
+ chown 8888:8888 /home/altuser
Removing intermediate container 9c7b30365c2d
 ---> fd7173d42923
Successfully built fd7173d42923
Successfully tagged customcortex:v2.x.x
```

That custom user can then be used to run the Cortex:

```
version: "3.3"
services:
  00.cortex:
    user: "8888"
    image: customcortex:v2.x.x
    network_mode: host
    restart: unless-stopped
    volumes:
    - ./storage:/vertex/storage
    environment:
    - SYN_CORTEX_AXON=aha://axon...
    - SYN_CORTEX_JSONSTOR=aha://jsonstor...
    - SYN_CORTEX_AHA_PROVISION=ssl://aha.<yournetwork>:27272/<guid>?certhash=<sha256>
```

The following bash script can be used to help automate this process, by adding the user to an image and appending the custom username to the image tag:

```
#!/bin/bash
# Add a user to a debian based container with an arbitrary uid/gid value.
# default username: altuser
# default uid: 8888

set -e

if [ -z $1 ]
then
    echo "Usage: srcImage name id suffix"
    echo "srcImage required."
    exit 1
fi

SRC_IMAGE_NAME=$1
NEW_NAME=${2:-altuser}
NEW_ID=${3:-8888}
SUFFIX=-${4:-$NEW_NAME}

echo "Add user/group ${NEW_NAME} with ${NEW_ID} into ${SRC_IMAGE_NAME}, creating: ${SRC_
↪IMAGE_NAME}${SUFFIX}"

printf "FROM $SRC_IMAGE_NAME \
\nRUN set -ex \\
    && groupadd -g $NEW_ID $NEW_NAME \\
    && useradd -r --home-dir=/home/$NEW_NAME -u $NEW_ID -g $NEW_NAME --shell /bin/bash
↪$NEW_NAME \\
    && mkdir -p /home/$NEW_NAME \\
    && chown $NEW_ID:$NEW_ID /home/$NEW_NAME\n" > ./Dockerfile

docker build -t $SRC_IMAGE_NAME$SUFFIX -f ./Dockerfile .

rm ./Dockerfile

exit 0
```

Saving this to `adduserimage.sh`, it can then be used to quickly modify an image. The following example shows running this to add a user named `foouser` with the uid 1234:

```
$ ./adduserimage.sh vertexproject/synapse-aha:v2.113.0 foouser 1234
Add user/group foouser with 1234 into vertexproject/synapse-aha:v2.113.0, creating:␣
↪vertexproject/synapse-aha:v2.113.0-foouser
Sending build context to Docker daemon  4.608kB
Step 1/2 : FROM vertexproject/synapse-aha:v2.113.0
 ---> 53251b832df0
Step 2/2 : RUN set -ex && groupadd -g 1234 foouser && useradd -r --home-dir=/home/
↪foouser -u 1234 -g foouser --shell /bin/bash foouser && mkdir -p /home/foouser &&␣
↪chown 1234:1234 /home/foouser
 ---> Running in 1c9e793d6761
+ groupadd -g 1234 foouser
+ useradd -r --home-dir=/home/foouser -u 1234 -g foouser --shell /bin/bash foouser
```

(continues on next page)

(continued from previous page)

```
+ mkdir -p /home/foouser
+ chown 1234:1234 /home/foouser
Removing intermediate container 1c9e793d6761
 ---> 21a12f395462
Successfully built 21a12f395462
Successfully tagged vertexproject/synapse-aha:v2.113.0-foouser
```

6.3 Synapse Services

6.3.1 AHA

The AHA service provides service discovery, provisioning, graceful mirror promotion, and certificate authority services to the other Synapse services. For a step-by-step guide to deploying an AHA instance, see the *Synapse Deployment Guide*. We will use `<yournetwork>` to specify locations where the value should be replaced with your chosen AHA network name.

Docker Image: `vertexproject/synapse-aha:v2.x.x`

Configuration

A typical AHA deployment requires some initial configuration options. At a minimum, you must specify the following:

```
aha:name: aha
aha:network: <yournetwork>
dmon:listen: ssl://aha.<yournetwork>&ca=<yournetwork>
```

To enable provisioning using AHA you must specify an alternate listener such as:

```
provision:listen: tcp://aha.<yournetwork>:27272
```

Note: The network connection from a Synapse service to the AHA service must NOT be passing through a Network Adress Translation (NAT) device.

For the full list supported options, see the *AHA Configuration Options*.

Using Aha with Custom Client Code

Loading the known AHA resolvers for use with custom python clients can be easily accomplished using the `withTeleEnv()` context manager:

```
import sys
import asyncio

import synapse.telepath as s_telepath

async def main(argv):

    # This context manager loads telepath.yaml
    async with s_telepath.withTeleEnv():

        async with await s_telepath.openurl(argv[0]) as proxy:
```

(continues on next page)

(continued from previous page)

```
            # call service provided telepath APIs

            info = await proxy.getCellInfo()
            print(repr(info))

    return 0

sys.exit(asyncio.run(main(sys.argv[1:])))
```

6.3.2 Axon

Note: If you are a Synapse Enterprise customer you should consider deploying the Synapse-S3 Axon.

The Axon service provides binary / blob storage inside of the Synapse ecosystem. Binary objects are indexed based on the SHA-256 hash so that storage of the same set of bytes is not duplicated. The Axon exposes a set of Telepath / HTTP APIs that can be used to upload, download, and check for the existence of a binary blob. For a step-by-step guide to deploying an Axon, see the *Synapse Deployment Guide*.

Docker Image: `vertexproject/synapse-axon:v2.x.x`

Note: For ease of use in simple deployments, the Cortex contains an embedded Axon instance. For production deployments it is **highly** recommended that you install it as a separated service to help distribute load and allow direct access by other Advanced Power-Ups.

Configuration

A typical Axon deployment does not require any additional configuration. For the full list supported options, see the *Axon Configuration Options*.

Permissions

axon
Controls access to all `axon.*` permissions.

axon.get
Controls access to retrieve a binary blob from the Axon based on the SHA256 hash.

axon.has
Controls access to check if bytes are present and return sizes based on the SHA256 hash.

axon.upload
Controls access to upload a binary blob to the Axon.

For example, to allow the user `visi` to upload, download, and confirm files you would execute the following command from **inside the Axon container**:

```
python -m synapse.tools.moduser --add visi --allow axon
```

6.3.3 JSONStor

The JSONStor is a utility service that provides a mechanism for storing and retrieving arbitrary JSON objects using a hierarchical naming system. It is commonly used to store user preferences, cache API query responses, and hold data that is not part of the *Data Model*. For an example of deploying a JSONStor, see the *Synapse Deployment Guide*.

Docker Image: `vertexproject/synapse-jsonstor:v2.x.x`

Note: For ease of use in simple deployments, the Cortex contains an embedded JSONStor instance. For production deployments it is **highly** recommended that you install it as a separated service to help distribute load and allow direct access by other Advanced Power-Ups.

Configuration

A typical JSONStor deployment does not require any additional configuration. For the full list supported options, see the *JSONStor Configuration Options*.

6.3.4 Cortex

A Cortex is the hypergraph database and main component of the Synapse service architecture. The Cortex is also where the Storm query language runtimes and execute where all automation and enrichment occurs. For a step-by-step guide to deploying a Cortex, see the *Synapse Deployment Guide*.

Docker Image: `vertexproject/synapse-cortex:v2.x.x`

Configuration

Many of the configurations and permissions managed within the Cortex are the responsibility of the global admin rather than the devops team. See the *Synapse Admin Guide* for details on global admin tasks and details.

The Cortex can be configured to log Storm queries executed by users. This is done by setting the `storm:log` and `storm:log:level` configuration options. The `storm:log:level` option may be one of `DEBUG`, `INFO` , `WARNING`, `ERROR`, `CRITICAL` This allows an organization to set what log level their Storm queries are logged at.

When enabled, the log message contains the query text and username:

```
2021-06-28 16:17:55,775 [INFO] Executing storm query {inet:ipv4=1.2.3.4} as [root]␣
↪[cortex.py:_logStormQuery:MainThread:MainProcess]
```

When structured logging is also enabled for a Cortex, the query text, username, and user iden are included as individual fields in the logged message as well:

```
{
  "message": "Executing storm query {inet:ipv4=1.2.3.4} as [root]",
  "logger": {
    "name": "synapse.storm",
    "process": "MainProcess",
    "filename": "cortex.py",
    "func": "_logStormQuery"
  },
  "level": "INFO",
  "time": "2021-06-28 16:18:47,232",
  "text": "inet:ipv4=1.2.3.4",
  "username": "root",
  "user": "3189065f95d3ab0a6904e604260c0be2"
}
```

This logging does interplay with the underlying log configuration (*Configure Logging*). The `storm:log:level` value must be greater than or equal to the `SYN_LOG_LEVEL`, otherwise the Storm log will not be emitted.

For the full list supported options, see the *Cortex Configuration Options*.

Extended HTTP API

The Cortex can be configured (via Storm) to service custom HTTP API endpoints. These user defined endpoints execute Storm code in order to generate responses. This allows creating custom HTTP API responses or URL paths which may meet custom needs.

These endpoints have a base URL of `/api/ext/`. Additional path components in a request are used to resolve which API definition is used to handle the response.

The Storm queries which implement these endpoints will have a `$request` object (see *http:api:request*) added to them. This object is used to send custom data back to the caller. This object contains helpers to access the request data, as well as functions to send data back to the caller.

Note: Several examples show curl and jq being used to access endpoints or process data. These tools are not required in order to interact with the Extended HTTP API.

A Simple Example

The following simple example shows adding an API endpoint and setting the `GET` method on it that just returns a simple message embedded in a dictionary:

```
$api = $lib.cortex.httpapi.add('demo/path00')
$api.methods.get = ${
    $mesg=`Hello! I am a request made to {$request.path}`
    $headers = ({"Some": "Value"})
    $body = ({"mesg": $mesg})
    $request.reply(200, headers=$headers, body=$body)
 }
```

When accessing that HTTP API endpoint on the Cortex, the response data has the status code, custom headers, and custom body in the reponse:

```
$ curl -D - -sku "root:root" "https://127.0.0.1:4443/api/ext/demo/path00"
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf8"
Date: Tue, 17 Oct 2023 16:21:32 GMT
Some: value
Content-Length: 53

{"mesg": "Hello! I am a request made to demo/path00"}
```

The `$request.reply()` method automatically will convert primitive objects into a JSON response, enabling rapid development of JSON based API endpoints.

Accessing Request Data

The `$request` object has information available about the request itself. The following API example shows access to all of that request data, and echoes it back to the caller:

```
$api = $lib.cortex.httpapi.add('demo/([a-z0-9]*)')
$api.methods.post = ${
    $body = ({
        "method": $request.method,        // The HTTP method
        "headers": $request.headers,      // Any request headers
        "params": $request.params,        // Any requets parameters
        "uri": $request.uri,              // The full URI requested
        "path": $request.path,            // The path component after /api/ext/
        "args": $request.args,            // Any capture groups matched from the path.
        "client": $request.client,        // Requester client IP
        "iden": $request.api.iden,        // The iden of the HTTP API handling the␣
↪request
        "nbyts": $lib.len($request.body), // The raw body is available as bytes
    })
    try {
        $body.json = $request.json        // Synapse will lazily load the request body␣
↪as json upon access
    } catch StormRuntimeError as err {    // But it may not be json!
        $body.json = 'err'
    }
    $headers = ({'Echo': 'hehe!'})
    $request.reply(200, headers=$headers, body=$body)
}
```

Accessing that endpoint shows that request information is echoed back to the caller:

```
$ curl -sku "root:secret" -XPOST -d '{"some":["json", "items"]}' "https://127.0.0.1:4443/
↪api/ext/demo/ohmy?hehe=haha" | jq
{
  "method": "POST",
  "headers": {
    "host": "127.0.0.1:4443",
    "authorization": "Basic cm9vdDpzZWNyZXQ=",
    "user-agent": "curl/7.81.0",
    "accept": "*/*",
    "content-length": "26",
    "content-type": "application/x-www-form-urlencoded"
  },
  "params": {
    "hehe": "haha"
  },
  "uri": "/api/ext/demo/ohmy?hehe=haha",
  "path": "demo/ohmy",
  "args": [
    "ohmy"
  ],
  "client": "127.0.0.1",
  "iden": "50cf80d0e332a31608331490cd453103",
```

(continues on next page)

(continued from previous page)

```
  "nbyts": 26,
  "json": {
    "some": [
      "json",
      "items"
    ]
  }
}
```

The `$request.headers` are accessed in a case-insensitive manner. `$request.parameters` are case sensitive. The following example shows that:

```
$api = $lib.cortex.httpapi.get(50cf80d0e332a31608331490cd453103)
$api.methods.get = ${
    $body=({
        "ua":   $request.headers."UseR-AGent",  // case insensitive match on the User-
↪Agent string
        "hehe": $request.params.hehe,
        "HEHE": $request.params.HEHE,
    })
    $request.reply(200, body=$body)
}
```

The output of that endpoint:

```
$ curl -s -k -u "root:secret" "https://127.0.0.1:4443/api/ext/demo/casemath?hehe=haha&
↪HEHE=uppercase"  | jq
{
  "ua": "curl/7.81.0",
  "hehe": "haha",
  "HEHE": "uppercase"
}
```

Note: Request headers and parameters are flattened into a single key / value mapping. Duplicate request headers or parameters are not exposed in the `$request` object.

Managing HTTP APIs

When creating an Extended HTTP API, the request path must be provided. This path component is matched against any path components after `/api/etx/*` when determing which API endpoint will service the request. The API endpoints are matched in order, comparing their `path` against the requested path using a case sensitive fullmatch regular expression comparison. Newly created API endpoints are added to the end of the list for matching. It is best for these endpoints to be ordered from most specific to least specific.

To list the registered APIs, their order, and path information, use the `cortex.httpapi.list` command:

```
storm> cortex.httpapi.list
order iden                             owner                auth  runas  path
0     50cf80d0e332a31608331490cd453103 root                 true  owner  demo/([a-z0-9]*)
1     586311d3a7a26d6138bdc07169e4cde5 root                 true  owner  demo/path00
```

(continues on next page)

(continued from previous page)

```
2     1896bda5dbd97615ee553059079620ba root                    true  owner  demo/path01
3     daaf33e23b16540acdc872fee2de1b61 root                    true  owner  something/Else
```

In this example, there are four items listed. The `path` of the first item will match the paths for the second and third items. The index for the first item needs to be moved using the `cortex.httpapi.index` commmand. That command allows users to change the order in which the API endpoints are matched:

```
storm> cortex.httpapi.index 50cf80d0e332a31608331490cd453103 3
Set HTTP API 50cf80d0e332a31608331490cd453103 to index 3

storm> cortex.httpapi.list
order iden                             owner                   auth  runas  path
0     586311d3a7a26d6138bdc07169e4cde5 root                    true  owner  demo/path00
1     1896bda5dbd97615ee553059079620ba root                    true  owner  demo/path01
2     daaf33e23b16540acdc872fee2de1b61 root                    true  owner  something/Else
3     50cf80d0e332a31608331490cd453103 root                    true  owner  demo/([a-z0-9]*)
```

The endpoints in the example are now checked in a "more specific" to "least specific" order.

The path of an endpoint can also be changed. This can be done by assigning a new value to the `path` attribute on the `http:api` object in Storm:

```
storm> $api=$lib.cortex.httpapi.get(1896bda5dbd97615ee553059079620ba) $api.path="demo/
↪mynew/path"
complete. 0 nodes in 8 ms (0/sec).

storm> cortex.httpapi.list
order iden                             owner                   auth  runas  path
0     586311d3a7a26d6138bdc07169e4cde5 root                    true  owner  demo/path00
1     1896bda5dbd97615ee553059079620ba root                    true  owner  demo/mynew/path
2     daaf33e23b16540acdc872fee2de1b61 root                    true  owner  something/Else
3     50cf80d0e332a31608331490cd453103 root                    true  owner  demo/([a-z0-9]*)
```

The path components which match each regular expression capture group in the `path` will be set in the `$request.args` data. An endpoint can capture multiple args this way:

```
// Set the echo API handler defined earlier to have a path which has multiple capture␣
↪groups
$api = $lib.cortex.httpapi.get(50cf80d0e332a31608331490cd453103)
$api.path="demo/([a-z0-9]+)/(.*)"
```

The capture groups are then available:

```
$ curl -sku "root:secret" -XPOST "https://127.0.0.1:4443/api/ext/demo/foobar1/
↪AnotherArgument/inTheGroup"  | jq '.args'
[
  "foobar1",
  "AnotherArgument/inTheGroup"
]
```

Note: The Cortex does not make any attempt to do any inspection of path values which may conflict between the endpoints. This is because the paths for a given endpoint may be changed, they can contain regular expressions, and they may have their resolution order changed. Cortex users are responsible for configuring their endpoints with correct paths and order to meet their use cases.

The Extended HTTP APIs can also be given a name and a description. The following shows setting the `name` and `desc` fields, and then showing the details of the API using `cortex.httpapi.stat`. This command shows detailed information about the Extended HTTP API endpoint:

```
$api = $lib.cortex.httpapi.get(50cf80d0e332a31608331490cd453103)
$api.name="demo wildcard"
$api.desc='''This API endpoint is a wildcard example. It has a GET method and a POST␣
↪method available.'''

// Stat output
storm> cortex.httpapi.stat 50cf80d0e332a31608331490cd453103
Iden: 50cf80d0e332a31608331490cd453103
Creator: root (b13c21813628ac4464b78b5d7c55cd64)
Created: 2023/10/18 14:02:52.070
Updated: 2023/10/18 14:07:29.448
Path: demo/([a-z0-9]+)/(.*)
Owner: root (b13c21813628ac4464b78b5d7c55cd64)
Runas: owner
View: default (a1877dd028915d90862e35e24b491bfc)
Readonly: false
Authenticated: true
Name: demo wildcard
Description: This API endpoint is a wildcard example. It has a GET method and a POST␣
↪method available.

No user permissions are required to run this HTTP API endpoint.
The handler defines the following HTTP methods:
Method: POST
$body = ({
            "method": $request.method,        // The HTTP method
            "headers": $request.headers,      // Any request headers
            "params": $request.params,        // Any requets parameters
            "uri": $request.uri,              // The full URI requested
            "path": $request.path,            // The path component after /api/ext/
            "args": $request.args,            // Any capture groups matched from the␣
↪path.
            "client": $request.client,        // Requester client IP
            "iden": $request.api.iden,        // The iden of the HTTP API handling the␣
↪request
            "nbyts": $lib.len($request.body), // The raw body is available as bytes
        })
        try {
            $body.json = $request.json        // Synapse will lazily load the request␣
↪body as json upon access
        } catch StormRuntimeError as err {    // But it may not be json!
            $body.json = 'err'
        }
        $headers = ({'Echo': 'hehe!'})
        $request.reply(200, headers=$headers, body=$body)
```

(continues on next page)

(continued from previous page)

```
Method: GET
$body=({
            "ua":   $request.headers."UseR-AGent",  // case insensitive match on the␣
↪User-Agent string
            "hehe": $request.params.hehe,
            "HEHE": $request.params.HEHE,
        })
        $request.reply(200, body=$body)

No vars are set for the handler.
```

Supported Methods

The endpoints support the following HTTP Methods:

- GET
- PUT
- HEAD
- POST
- PATCH
- DELETE
- OPTIONS

The logic which implements these methods is set via Storm. The following example shows setting two simple methods for a given endpoint:

```
$api = $lib.cortex.httpapi.get(586311d3a7a26d6138bdc07169e4cde5)
$api.methods.get = ${ $request.reply(200, headers=({"X-Method": "GET"})) 
$api.methods.put = ${ $request.reply(200, headers=({"X-Method": "PUT"}))
```

These methods can be removed as well by assigning `$lib.undef` to the value:

```
// Remove the GET method
$api = $lib.cortex.httpapi.get(586311d3a7a26d6138bdc07169e4cde5)
$api.methods.put = $lib.undef
```

Users are not required to implement their methods in any particular styles or conventions. The only method specific restriction on the endpoint logic is for the `HEAD` method. Any body content that is sent in response to the `HEAD` method will not be transmitted to the requester. This body content will be omitted from being transmitted without warning or error.

A request which is made with for method that a matching handler does not implement will return an HTTP 405 error.

Authentication, Permissions, and Users

Since the endpoints are executed by running Storm queries to generate responses, Synapse must resolve the associated *User* and a *View* which will be used to run the query. There are a few important properties of the endpoints that users configuring them must be aware of.

owner

By default, the user that creates an endpoint is marked as the `owner` for that endpoint. This is the default user that will execute the Storm queries which implement the HTTP Methods. This value can be changed by setting the `.owner` property on the endpoint object to a different User.

A user marked as the `owner` of an endpoint does not have any permissions granted that allows them to edit the endpoint.

view

The View that an Extended HTTP API endpoint is created in is recorded as the View that the Storm endpoints are executed in. This View can be changed by assigning the `.view` property on the endpoint object to a different View.

authenticated

By default, the endpoints require the requester to have an authenticated session. Information about API authentication can be found at *Authentication*. This authentication requirement can be disabled by setting the `.authenticated` property on the endpoint object to `$lib.false`. That will allow the endpoint to be resolved without presenting any sort of authentication information.

runas

By default, the Storm logic is run by the user that is marked as the `owner`. Endpoints can instead be configured to run as the authenticated user by setting the `.runas` property on the HTTP API object to `user`. In order to change the behavior to executing the queries as the owner, the value should be set to `owner`.

When an endpoint is configured with `runas` set to `user` and `authenticated` to `$lib.false` any calls to that API will be executed as the `owner`.

This allows creating endpoints that run in one of three modes:

- Authenticated & runs as the Owner
- Authenticated & runs as the User
- Unauthenticated & runs as the Owner

These three modes can be demonstrated by configuring endpoints that will echo back the current user:

```
// Create a query object that we will use for each handler
$echo=${ $request.reply(200, body=$lib.user.name()) }

// Create the first endpoint with a default configuration.
$api0 = $lib.cortex.httpapi.add('demo/owner')
$api0.methods.get=$echo

// Create the second endpoint which runs its logic as the requester.
$api1 = $lib.cortex.httpapi.add('demo/user')
$api1.runas=user
$api1.methods.get=$echo
```

(continues on next page)

(continued from previous page)

```
// Create the third endpoint which does not require authentication.
$api2 = $lib.cortex.httpapi.add('demo/noauth')
$api2.authenticated=$lib.false  // Disable authentication
$api2.methods.get=$echo
```

Accessing those endpoints with different users gives various results:

```
# The demo/owner endpoint runs as the owner
$ curl -sku "root:secret" "https://127.0.0.1:4443/api/ext/demo/owner"  | jq
"root"

$ curl -sku "lowuser:demo" "https://127.0.0.1:4443/api/ext/demo/owner"  | jq
"root"

# The demo/user endpoint runs as the requester
$ curl -sku "root:secret" "https://127.0.0.1:4443/api/ext/demo/user"  | jq
"root"

$ curl -sku "lowuser:demo" "https://127.0.0.1:4443/api/ext/demo/user"  | jq
"lowuser"

# The demo/noauth endpoint runas the owner
$ curl -sk "https://127.0.0.1:4443/api/ext/demo/noauth"  | jq
"root"
```

If the owner or an authenticated user does not have permission to execute a Storm query in the configured View, or if the endpoints' View is deleted from the Cortex, this will raise a fatal error and return an HTTP 500 error. Once a query has started executing, regular Storm permissions apply.

Endpoints can also have permissions defined for them. This allows locking down an endpoint such that while a user may still have access to the underlying view, they may lack the specific permissions required to execute the endpoint. These permissions are checked against the authenticated user, and not the endpoint owner. The following example shows setting a single permission on one of our earlier endpoints:

```
$api=$lib.cortex.httpapi.get(bd4679ab8e8a1fbc030b46e275ddba96)
$api.perms=(your.custom.permission,)
```

Accessing it as a user without the specified permission generates an `AuthDeny` error:

```
$ curl -sku "lowuser:demo" "https://127.0.0.1:4443/api/ext/demo/owner"  | jq
{
  "status": "err",
  "code": "AuthDeny",
  "mesg": "User (lowuser) must have permission your.custom.permission"
}
```

The user can have that permission granted via Storm:

```
storm> auth.user.addrule lowuser your.custom.permission
Added rule your.custom.permission to user lowuser.
```

Then the endpoint can be accessed:

```
$ curl -sku "lowuser:demo" "https://127.0.0.1:4443/api/ext/demo/owner"  | jq
"root"
```

For additional information about managing user permissions, see *Create and Manage Users and Roles*.

Note: When the Optic UI is used to proxy the `/api/ext` endpoint, authentication must be done using Optic's login endpoint. Basic auth is not available.

Readonly Mode

The Storm queries for a given handler may be executed in a `readonly` runtime. This is disabled by default. This can be changed by setting the `readonly` attribute on the `http:api` object:

```
// Enable the Storm queries to be readonly
$api = $lib.cortex.httpapi.get($yourIden)
$api.readonly = $lib.true
```

Endpoint Variables

User defined variables may be set for the queries as well. These variables are mapped into the runtime for each method. This can be used to provide constants or other information which may change, without needing to alter the underlying Storm code which defines a method. These can be read (or removed) by altering the `$api.vars` dictionary. This is an example of using a variable in a query:

```
// Set a variable that a method uses:

$api = $lib.cortex.httpapi.get($yourIden)
$api.methods.get = ${
    $mesg = `There are {$number} things available!`
    $request.reply(200, body=({"mesg": $mesg})
}
$api.vars.number = (5)
```

When executing this method, the JSON response would be the following:

```
{"mesg": "There are 5 things available!"}
```

If `$api.vars.number = "several"` was executed, the JSON response would now be the following:

```
{"mesg": "There are several things available!"}
```

Variables can be removed by assigning `$lib.undef` to them:

```
$api = $lib.cortex.httpapi.get($yourIden)
$api.vars.number = $lib.undef
```

Sending Custom Responses

Responses can be made which are not JSON formatted. The $request.reply() method can be used to send raw bytes. The user must provide any appropriate headers alongside their request.

HTML Example

The following example shows an endpoint which generates a small amount of HTML. It uses an HTML template stored in in the method vars. This template has a small string formatted in it, converted to bytes, and then the headers are set. The end result can be then rendered in a web browser:

```
$api = $lib.cortex.httpapi.add('demo/html')
$api.vars.template = '''<!DOCTYPE html>
<html>
<body>
<h1>A Header</h1>
<p>{mesg}</p>
</body>
</html>'''
$api.methods.get = ${
    $duration = $lib.model.type(duration).repr($lib.cell.uptime().uptime)
    $mesg = `The Cortex has been up for {$duration}`
    $html = $lib.str.format($template, mesg=$mesg)
    $buf = $html.encode()
    $headers = ({
        "Content-Type": "text/html",
        "Content-Length": `{$lib.len($buf)}`
    })
    $request.reply(200, headers=$headers, body=$buf)
}
```

Accessing this endpoint with curl shows the following:

```
$ curl -D - -sku "root:secret" "https://127.0.0.1:4443/api/ext/demo/html"
HTTP/1.1 200 OK
Content-Type: text/html
Date: Wed, 18 Oct 2023 14:07:47 GMT
Content-Length: 137

<!DOCTYPE html>
<html>
<body>
<h1>A Header</h1>
<p>The Cortex has been up for 1D 00:59:12.704</p>
</body>
</html>f
```

Streaming Examples

The http:request object has methods that allow a user to send the response code, headers and body separately. One use for this is to create a streaming response. This can be used when the total response size may not be known or to avoid incurring memory pressure on the Cortex when computing results.

The following examples generates some JSONLines data:

```
$api = $lib.cortex.httpapi.add('demo/jsonlines')
$api.methods.get = ${
    $request.sendcode(200)
    // This allows a browser to view the response
    $request.sendheaders(({"Content-Type": "text/plain; charset=utf8"}))
    $values = ((1), (2), (3))
    for $i in $values {
        $data = ({'i': $i})
        $body=`{$lib.json.save($data)}\n`
        $request.sendbody($body.encode())
    }
}
```

Accessing this endpoint shows the JSONLines rows sent back:

```
$ curl -D - -sku "root:secret" "https://127.0.0.1:4443/api/ext/demo/jsonlines"
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf8
Date: Wed, 18 Oct 2023 14:31:29 GMT
 nosniff
Transfer-Encoding: chunked

{"i": 1}
{"i": 2}
{"i": 3}
```

In a similar fashion, a CSV can be generated. This example shows an integer and its square being computed:

```
$api = $lib.cortex.httpapi.add('demo/csv')
$api.methods.get = ${
    $request.sendcode(200)
    $request.sendheaders(({"Content-Type": "text/csv"}))

    // Header row
    $header="i, square\n"
    $request.sendbody($header.encode())

    $n = 10  // Number of rows to compute
    for $i in $lib.range($n) {
        $square = ($i * $i)
        $body = `{$i}, {$square}\n`
        $request.sendbody($body.encode())
    }
}
```

Accessing this shows the CSV content being sent back:

```
$ curl -D - -sku "root:secret" "https://127.0.0.1:4443/api/ext/demo/csv"
HTTP/1.1 200 OK
Content-Type: text/csv
Date: Wed, 18 Oct 2023 14:43:37 GMT
Transfer-Encoding: chunked

```

(continues on next page)

(continued from previous page)

```
i, square
0, 0
1, 1
2, 4
3, 9
4, 16
5, 25
6, 36
7, 49
8, 64
9, 81
```

When using the `sendcode()`, `sendheaders()`, and `sendbody()` APIs the order in which they are called does matter. The status code and headers can be set at any point before sending body data. They can even be set multiple times if the response logic needs to change a value it previously set.

Once the body data has been sent, the status code and headers will be sent to the HTTP client and cannot be changed. Attempting to change the status code or send additional headers will have no effect. This will generate a warning message on the Cortex.

The **minimum** data that the Extended HTTP API requires for a response to be considered valid is setting the status code. If the status code is not set by an endpoint, or if body content is sent prior to setting the endpoint, then an HTTP 500 status code will be sent to the caller.

Messages and Error Handling

Messages sent out of the Storm runtime using functions such as `$lib.print()`, `$lib.warn()`, or `$lib.fire()` are not available to HTTP API callers. The `$lib.log` Storm library can be used for doing out of band logging of messages that need to be generated while handling a response.

A Storm query which generates an error which tears down the Storm runtime with an `err` message will result in an HTTP 500 response being sent. The error will be encoded in the Synapse HTTP API error convention documented at *HTTP/REST API Conventions*.

For example, if the previous example where the handler sent a `mesg` about the `$number` of things available was run after the variable `$number` was removed, the code would generate the following response body:

```
{"status": "err", "code": "NoSuchVar", "mesg": "Missing variable: number"}
```

Custom error handling of issues that arise inside of the Storm query execution can be handled with the *Try...Catch Statement*. This allows a user to have finer control over their error codes, headers and error body content.

Note: The HTTP 500 response will not be sent if there has already been body data send by the endpoint.

6.4 Devops Details

6.4.1 Orchestration

Kubernetes

A popular option for Orchestration is Kubernetes. Kubernetes is an open-source system for automating the deployment, scaling and management of containerized applications. Synapse does work in Kubernetes environments.

Note: If you are using these examples to get started with Synapse on Kubernetes, you may need to adapt them to meet operational needs for your environment.

Example Deployment

The following examples walk through deploying an example Synapse deployment (based on *Synapse Deployment Guide*), but inside of a managed Kubernetes cluster managed by Digital Ocean. This deployment makes a few assumptions:

Synapse Deployment Guide
This guide assumes a familiarity with the Synapse deployment guide. Concepts covered there are not repeated here.

namespace
These examples use the Kubernetes `default` namespace.

PersistentVolumeClaim
These examples use PersistentVolumeClaim (PVC) to create a persistent storage location. All Synapse services assume they have some persistent storage to read and write to. This example uses the `storageClass` of `do-block-storage`. You may need to alter these examples to provide a `storageClass` that is appropriate for your environment.

Aha naming
In Kubernetes, we rely on the default naming behavior for services to find the Aha service via DNS, so our Aha name and Aha network should match the internal naming for services in the cluster. The `aha:network` value is `<namespace>.<cluster dns root>`. This DNS root value is normally `svc.cluster.local`, so the resulting DNS label for the Aha service is `aha.default.svc.cluster.local`. Similarly, the Aha service is configured to listen on `0.0.0.0`, since we cannot bind the DNS label provided by Kubernetes prior to the Pod running Aha being available.

Aha

The following `aha.yaml` can be used to deploy an Aha service.

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-aha
  labels:
    app.kubernetes.io/name: "aha"
    app.kubernetes.io/instance: "aha"
```

(continues on next page)

(continued from previous page)

```
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "aha"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: do-block-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aha
  labels:
    app.kubernetes.io/name: "aha"
    app.kubernetes.io/instance: "aha"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "aha"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "aha"
      app.kubernetes.io/instance: "aha"
      app.kubernetes.io/version: "v2.x.x"
      app.kubernetes.io/component: "aha"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "aha"
        app.kubernetes.io/instance: "aha"
        app.kubernetes.io/version: "v2.x.x"
        app.kubernetes.io/component: "aha"
        app.kubernetes.io/part-of: "synapse"
        environment: "dev"
    spec:
      securityContext:
        runAsUser: 999
        runAsGroup: 999
        fsGroup: 999
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: example-aha
```

(continues on next page)

(continued from previous page)

```
      containers:
      - name: aha
        image: vertexproject/synapse-aha:v2.x.x
        env:
          - name: SYN_LOG_LEVEL
            value: DEBUG
          - name: SYN_LOG_STRUCT
            value: "false"
          - name: SYN_AHA_AHA_NAME
            value: aha
          - name: SYN_AHA_AHA_NETWORK
            # This is <namespace>.<cluster dns root> - it is used as Certificate␣
↪Authority name
            value: default.svc.cluster.local
          - name: SYN_AHA_DMON_LISTEN
            # This is <aha name>.<namespace>.<cluster dns root> and the CA name from␣
↪above
            value: "ssl://0.0.0.0?hostname=aha.default.svc.cluster.local&ca=default.svc.
↪cluster.local"
          - name: SYN_AHA_PROVISION_LISTEN
            # This is <aha name>.<namespace>.<cluster dns root>
            value: "ssl://0.0.0.0:27272?hostname=aha.default.svc.cluster.local"
          - name: SYN_AHA_HTTPS_PORT
            value: null
        volumeMounts:
          - mountPath: /vertex/storage
            name: data
        imagePullPolicy: Always
        startupProbe:
          failureThreshold: 2147483647
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
        readinessProbe:
          failureThreshold: 2
          initialDelaySeconds: 20
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: aha
  labels:
    app.kubernetes.io/name: "aha"
    app.kubernetes.io/instance: "aha"
```

(continues on next page)

(continued from previous page)

```
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "aha"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  type: ClusterIP
  selector:
    app.kubernetes.io/instance: aha
    environment: "dev"
  ports:
    - port: 27492
      protocol: TCP
      name: telepath
    - port: 27272
      protocol: TCP
      name: provisioning
```

This can be deployed via `kubectl apply`. That will create the PVC, deployment, and service.

```
$ kubectl apply -f aha.yaml
persistentvolumeclaim/example-aha created
deployment.apps/aha created
service/aha created
```

You can see the startup logs as well:

```
$ kubectl logs -l app.kubernetes.io/instance=aha
2023-03-08 04:22:02,568 [DEBUG] Set config valu from envar: [SYN_AHA_DMON_LISTEN]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 04:22:02,568 [DEBUG] Set config valu from envar: [SYN_AHA_HTTPS_PORT] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 04:22:02,568 [DEBUG] Set config valu from envar: [SYN_AHA_AHA_NAME] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 04:22:02,569 [DEBUG] Set config valu from envar: [SYN_AHA_AHA_NETWORK]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 04:22:02,651 [INFO] Adding CA certificate for default.svc.cluster.local [aha.
↪py:initServiceRuntime:MainThread:MainProcess]
2023-03-08 04:22:02,651 [INFO] Generating CA certificate for default.svc.cluster.local␣
↪[aha.py:genCaCert:MainThread:MainProcess]
2023-03-08 04:22:06,401 [INFO] Adding server certificate for aha.default.svc.cluster.
↪local [aha.py:initServiceRuntime:MainThread:MainProcess]
2023-03-08 04:22:08,879 [INFO] dmon listening: ssl://0.0.0.0?hostname=aha.default.svc.
↪cluster.local&ca=default.svc.cluster.local [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
2023-03-08 04:22:08,882 [INFO] ...ahacell API (telepath): ssl://0.0.0.0?hostname=aha.
↪default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 04:22:08,882 [INFO] ...ahacell API (https): disabled [cell.
↪py:initFromArgv:MainThread:MainProcess]
```

Axon

The following axon.yaml can be used as the basis to deploy an Axon service.

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-axon00
  labels:
    app.kubernetes.io/name: "axon"
    app.kubernetes.io/instance: "axon00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "axon"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: do-block-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: axon00
  labels:
    app.kubernetes.io/name: "axon"
    app.kubernetes.io/instance: "axon00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "axon"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "axon"
      app.kubernetes.io/instance: "axon00"
      app.kubernetes.io/version: "v2.x.x"
      app.kubernetes.io/component: "axon"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "axon"
        app.kubernetes.io/instance: "axon00"
        app.kubernetes.io/version: "v2.x.x"
        app.kubernetes.io/component: "axon"
```

(continues on next page)

(continued from previous page)

```
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  spec:
    securityContext:
      runAsUser: 999
      runAsGroup: 999
      fsGroup: 999
    volumes:
      - name: data
        persistentVolumeClaim:
          claimName: example-axon00
    containers:
    - name: axon
      image: vertexproject/synapse-axon:v2.x.x
      env:
        - name: SYN_LOG_LEVEL
          value: DEBUG
        - name: SYN_LOG_STRUCT
          value: "false"
        - name: SYN_AXON_AHA_PROVISION
          value: "ssl://aha.default.svc.cluster.local:27272/..."
        - name: SYN_AXON_HTTPS_PORT
          value: null
      volumeMounts:
        - mountPath: /vertex/storage
          name: data
      imagePullPolicy: Always
      startupProbe:
        failureThreshold: 2147483647
        timeoutSeconds: 20
        periodSeconds: 20
        exec:
          command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
      readinessProbe:
        failureThreshold: 2
        initialDelaySeconds: 20
        timeoutSeconds: 20
        periodSeconds: 20
        exec:
          command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
    restartPolicy: Always
```

Before we deploy that, we need to create the Aha provisioning URL. We can do that via `kubectl exec`. That should look like the following:

```
$ kubectl exec deployment/aha -- python -m synapse.tools.aha.provision.service 00.axon
one-time use URL: ssl://aha.default.svc.cluster.local:27272/
↪39a33f6e3fa2b512552c2c7770e28d30?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f
```

We want to copy that URL into the `SYN_AXON_AHA_PROVISION` environment variable, so that block looks like the following:

```
- name: SYN_AXON_AHA_PROVISION
  value: "ssl://aha.default.svc.cluster.local:27272/39a33f6e3fa2b512552c2c7770e28d30?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f"
```

This can then be deployed via `kubectl apply`:

```
$ kubectl apply -f axon.yaml
persistentvolumeclaim/example-axon00 unchanged
deployment.apps/axon00 created
```

You can see the Axon logs as well. These show provisioning and listening for traffic:

```
$ kubectl logs -l app.kubernetes.io/instance=axon00
2023-03-08 17:27:44,721 [INFO] log level set to DEBUG [common.
↪py:setlogging:MainThread:MainProcess]
2023-03-08 17:27:44,722 [DEBUG] Set config valu from envar: [SYN_AXON_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:27:44,722 [DEBUG] Set config valu from envar: [SYN_AXON_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:27:44,723 [INFO] Provisioning axon from AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:27:44,833 [DEBUG] Set config valu from envar: [SYN_AXON_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:27:44,833 [DEBUG] Set config valu from envar: [SYN_AXON_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:27:51,649 [INFO] Done provisioning axon AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:27:51,898 [INFO] dmon listening: ssl://0.0.0.0:0?hostname=00.axon.default.
↪svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
2023-03-08 17:27:51,899 [INFO] ...axon API (telepath): ssl://0.0.0.0:0?hostname=00.axon.
↪default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 17:27:51,899 [INFO] ...axon API (https): disabled [cell.
↪py:initFromArgv:MainThread:MainProcess]
```

The hostname `00.axon.default.svc.cluster.local` seen in the logs is **not** a DNS label in Kubernetes. That is an internal label used by the service to resolve SSL certificates that it provisioned with the Aha service, and as the name that it uses to register with the Aha service.

JSONStor

The following `jsonstor.yaml` can be used as the basis to deploy a JSONStor service.

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-jsonstor00
  labels:
    app.kubernetes.io/name: "jsonstor"
```

(continues on next page)

(continued from previous page)

```yaml
    app.kubernetes.io/instance: "jsonstor00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "jsonstor"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: do-block-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jsonstor00
  labels:
    app.kubernetes.io/name: "jsonstor"
    app.kubernetes.io/instance: "jsonstor00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "jsonstor"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "jsonstor"
      app.kubernetes.io/instance: "jsonstor00"
      app.kubernetes.io/version: "v2.x.x"
      app.kubernetes.io/component: "jsonstor"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "jsonstor"
        app.kubernetes.io/instance: "jsonstor00"
        app.kubernetes.io/version: "v2.x.x"
        app.kubernetes.io/component: "jsonstor"
        app.kubernetes.io/part-of: "synapse"
        environment: "dev"
    spec:
      securityContext:
        runAsUser: 999
        runAsGroup: 999
        fsGroup: 999
      volumes:
        - name: data
          persistentVolumeClaim:
```

(continues on next page)

(continued from previous page)

```
          claimName: example-jsonstor00
      containers:
      - name: jsonstor
        image: vertexproject/synapse-jsonstor:v2.x.x
        env:
          - name: SYN_LOG_LEVEL
            value: DEBUG
          - name: SYN_LOG_STRUCT
            value: "false"
          - name: SYN_JSONSTOR_AHA_PROVISION
            value: "ssl://aha.default.svc.cluster.local:27272/..."
          - name: SYN_JSONSTOR_HTTPS_PORT
            value: null
        volumeMounts:
          - mountPath: /vertex/storage
            name: data
        imagePullPolicy: Always
        startupProbe:
          failureThreshold: 2147483647
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
        readinessProbe:
          failureThreshold: 2
          initialDelaySeconds: 20
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
      restartPolicy: Always
```

Before we deploy that, we need to create the Aha provisioning URL. We can do that via `kubectl exec`. That should look like the following:

```
$ kubectl exec deployment/aha -- python -m synapse.tools.aha.provision.service 00.
↪jsonstor
one-time use URL: ssl://aha.default.svc.cluster.local:27272/
↪cbe50bb470ba55a5df9287391f843580?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f
```

We want to copy that URL into the `SYN_JSONSTOR_AHA_PROVISION` environment variable, so that block looks like the following:

```
- name: SYN_JSONSTOR_AHA_PROVISION
  value: "ssl://aha.default.svc.cluster.local:27272/cbe50bb470ba55a5df9287391f843580?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f"
```

This can then be deployed via `kubectl apply`:

```
$ kubectl apply -f jsonstor.yaml
persistentvolumeclaim/example-jsonstor00 created
deployment.apps/jsonstor00 created
```

You can see the JSONStor logs as well. These show provisioning and listening for traffic:

```
$ kubectl logs -l app.kubernetes.io/instance=jsonstor00
2023-03-08 17:29:15,137 [INFO] log level set to DEBUG [common.
↪py:setlogging:MainThread:MainProcess]
2023-03-08 17:29:15,137 [DEBUG] Set config valu from envar: [SYN_JSONSTOR_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:15,138 [DEBUG] Set config valu from envar: [SYN_JSONSTOR_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:15,140 [INFO] Provisioning jsonstorcell from AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:29:15,261 [DEBUG] Set config valu from envar: [SYN_JSONSTOR_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:15,261 [DEBUG] Set config valu from envar: [SYN_JSONSTOR_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:19,325 [INFO] Done provisioning jsonstorcell AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:29:19,966 [INFO] dmon listening: ssl://0.0.0.0:0?hostname=00.jsonstor.
↪default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
2023-03-08 17:29:19,966 [INFO] ...jsonstorcell API (telepath): ssl://0.0.0.0:0?
↪hostname=00.jsonstor.default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 17:29:19,966 [INFO] ...jsonstorcell API (https): disabled [cell.
↪py:initFromArgv:MainThread:MainProcess]
```

Cortex

The following `cortex.yaml` can be used as the basis to deploy the Cortex.

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-cortex00
  labels:
    app.kubernetes.io/name: "cortex"
    app.kubernetes.io/instance: "cortex00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "cortex"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

(continues on next page)

(continued from previous page)

```
  storageClassName: do-block-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cortex00
  labels:
    app.kubernetes.io/name: "cortex"
    app.kubernetes.io/instance: "cortex00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "cortex"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "cortex"
      app.kubernetes.io/instance: "cortex00"
      app.kubernetes.io/version: "v2.x.x"
      app.kubernetes.io/component: "cortex"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "cortex"
        app.kubernetes.io/instance: "cortex00"
        app.kubernetes.io/version: "v2.x.x"
        app.kubernetes.io/component: "cortex"
        app.kubernetes.io/part-of: "synapse"
        environment: "dev"
    spec:
      securityContext:
        runAsUser: 999
        runAsGroup: 999
        fsGroup: 999
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: example-cortex00
      containers:
      - name: cortex
        image: vertexproject/synapse-cortex:v2.x.x
        env:
          - name: SYN_LOG_LEVEL
            value: DEBUG
          - name: SYN_LOG_STRUCT
            value: "false"
          - name: SYN_CORTEX_AHA_PROVISION
            value: "ssl://aha.default.svc.cluster.local:27272/..."
```

(continues on next page)

(continued from previous page)

```
          - name: SYN_CORTEX_HTTPS_PORT
            value: null
          - name: SYN_CORTEX_STORM_LOG
            value: "true"
          - name: SYN_CORTEX_JSONSTOR
            value: "aha://jsonstor..."
          - name: SYN_CORTEX_AXON
            value: "aha://axon..."
        volumeMounts:
          - mountPath: /vertex/storage
            name: data
        imagePullPolicy: Always
        startupProbe:
          failureThreshold: 2147483647
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
        readinessProbe:
          failureThreshold: 2
          initialDelaySeconds: 20
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: cortex
  labels:
    app.kubernetes.io/name: "cortex"
    app.kubernetes.io/instance: "cortex00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "cortex"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  type: ClusterIP
  selector:
    app.kubernetes.io/instance: cortex00
    environment: "dev"
  ports:
    - port: 27492
      protocol: TCP
      name: telepath
```

Before we deploy that, we need to create the Aha provisioning URL. This uses a fixed listening port for the Cortex, so that we can later use port-forwarding to access the Cortex service. We do this via `kubectl exec`. That should look like the following:

```
$ kubectl exec deployment/aha -- python -m synapse.tools.aha.provision.service 00.cortex␣
↪--dmon-port 27492
one-time use URL: ssl://aha.default.svc.cluster.local:27272/
↪c06cd588e469a3b7f8a56d98414acf8a?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f
```

We want to copy that URL into the `SYN_CORTEX_AHA_PROVISION` environment variable, so that block looks like the following:

```
- name: SYN_CORTEX_AHA_PROVISION
  value: "ssl://aha.default.svc.cluster.local:27272/c06cd588e469a3b7f8a56d98414acf8a?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f"
```

This can then be deployed via `kubectl apply`:

```
$ kubectl apply -f cortex.yaml
persistentvolumeclaim/example-cortex00 created
deployment.apps/cortex00 created
service/cortex created
```

You can see the Cortex logs as well. These show provisioning and listening for traffic, as well as the connection being made to the Axon and JSONStor services:

```
$ kubectl logs -l app.kubernetes.io/instance=cortex00
2023-03-08 17:29:16,892 [INFO] log level set to DEBUG [common.
↪py:setlogging:MainThread:MainProcess]
2023-03-08 17:29:16,893 [DEBUG] Set config valu from envar: [SYN_CORTEX_AXON] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:16,893 [DEBUG] Set config valu from envar: [SYN_CORTEX_JSONSTOR]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:16,894 [DEBUG] Set config valu from envar: [SYN_CORTEX_STORM_LOG]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:16,894 [DEBUG] Set config valu from envar: [SYN_CORTEX_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:16,894 [DEBUG] Set config valu from envar: [SYN_CORTEX_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:16,896 [INFO] Provisioning cortex from AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:29:17,008 [DEBUG] Set config valu from envar: [SYN_CORTEX_AXON] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:17,009 [DEBUG] Set config valu from envar: [SYN_CORTEX_JSONSTOR]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:17,009 [DEBUG] Set config valu from envar: [SYN_CORTEX_STORM_LOG]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:17,010 [DEBUG] Set config valu from envar: [SYN_CORTEX_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:17,010 [DEBUG] Set config valu from envar: [SYN_CORTEX_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:29:20,356 [INFO] Done provisioning cortex AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:29:21,077 [INFO] dmon listening: ssl://0.0.0.0:27492?hostname=00.cortex.
↪default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
```

(continues on next page)

(continued from previous page)

```
2023-03-08 17:29:21,078 [INFO] ...cortex API (telepath): ssl://0.0.0.0:27492?hostname=00.
↪cortex.default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 17:29:21,078 [INFO] ...cortex API (https): disabled [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 17:29:21,082 [DEBUG] Connected to remote axon aha://axon... [cortex.
↪py:onlink:MainThread:MainProcess]
2023-03-08 17:29:21,174 [DEBUG] Connected to remote jsonstor aha://jsonstor... [cortex.
↪py:onlink:MainThread:MainProcess]
```

CLI Tooling Example

Synapse services and tooling assumes that IP and Port combinations registered with the AHA service are reachable. This example shows a way to connect to the Cortex from **outside** of the Kubernetes cluster without resolving service information via Aha. Communication between services inside of the cluster does not need to go through these steps. This does assume that your local environment has the Python synapse package available.

First add a user to the Cortex:

```
$ kubectl exec -it deployment/cortex00 -- python -m synapse.tools.moduser --add --admin␣
↪true visi
Adding user: visi
...setting admin: true
```

Then we need to generate a user provisioning URL:

```
$ kubectl exec -it deployment/aha -- python -m synapse.tools.aha.provision.user visi
one-time use URL: ssl://aha.default.svc.cluster.local:27272/
↪5d67f84c279afa240062d2f3b32fdb99?
↪certhash=e32d0e1da01b5eb0cefd4c107ddc8c8221a9a39bce25dea04f469c6474d84a23
```

Port-forward the AHA provisioning service to your local environment:

```
$ kubectl port-forward service/aha 27272:provisioning
```

Run the enroll tool to create a user certificate pair and have it signed by the Aha service. We replace the service DNS name of `aha.default.svc.cluster.local` with `localhost` in this example.

```
$ python -m synapse.tools.aha.enroll ssl://localhost:27272/
↪5d67f84c279afa240062d2f3b32fdb99?
↪certhash=e32d0e1da01b5eb0cefd4c107ddc8c8221a9a39bce25dea04f469c6474d84a23
Saved CA certificate: /home/visi/.syn/certs/cas/default.svc.cluster.local.crt
Saved user certificate: /home/visi/.syn/certs/users/visi@default.svc.cluster.local.crt
Updating known AHA servers
```

The Aha service port-forward can be disabled, and replaced with a port-forward for the Cortex service:

```
kubectl port-forward service/cortex 27492:telepath
```

Then connect to the Cortex via the Storm CLI, using the URL `ssl://visi@localhost:27492/?hostname=00.cortex.default.svc.cluster.local`.

```
$ python -m synapse.tools.storm "ssl://visi@localhost:27492/?hostname=00.cortex.default.
↪svc.cluster.local"

Welcome to the Storm interpreter!

Local interpreter (non-storm) commands may be executed with a ! prefix:
    Use !quit to exit.
    Use !help to see local interpreter commands.

storm>
```

The Storm CLI tool can then be used to run Storm commands.

Commercial Components

For Synapse-Enterprise users, deploying commercial components can follow a similar pattern. The following is an example of deploying Optic, the Synapse User Interface, as it is a common part of a Synapse deployment. This enables users to interact with Synapse via a web browser, instead of using the CLI tools. This example shows accessing the service via a port-forward. This example does not contain the full configuration settings you will need for a production deployment of Optic, please see *Synapse User Interface* for more information.

Note: Optic is available as a part of the **Synapse Enterprise** commercial offering. This example assumes that the Kubernetes cluster has a valid `imagePullSecret` named `regcred` which can access commercial images.

The following `optic.yaml` can be used as the basis to deploy Optic.

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-optic00
  labels:
    app.kubernetes.io/name: "optic"
    app.kubernetes.io/instance: "optic00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "optic"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  # You will need to use an appropriate storageClassName for your cluster.
  storageClassName: do-block-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: optic00
```

(continues on next page)

(continued from previous page)

```
  labels:
    app.kubernetes.io/name: "optic"
    app.kubernetes.io/instance: "optic00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "optic"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "optic"
      app.kubernetes.io/instance: "optic00"
      app.kubernetes.io/version: "v2.x.x"
      app.kubernetes.io/component: "optic"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "optic"
        app.kubernetes.io/instance: "optic00"
        app.kubernetes.io/version: "v2.x.x"
        app.kubernetes.io/component: "optic"
        app.kubernetes.io/part-of: "synapse"
        environment: "dev"
    spec:
      securityContext:
        runAsUser: 999
        runAsGroup: 999
        fsGroup: 999
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: example-optic00
      containers:
      - name: optic
        image: vertexproject/optic:v2.x.x
        securityContext:
          readOnlyRootFilesystem: true
        env:
          - name: SYN_LOG_LEVEL
            value: DEBUG
          - name: SYN_LOG_STRUCT
            value: "false"
          - name: SYN_OPTIC_AHA_PROVISION
            value: "ssl://aha.default.svc.cluster.local:27272/..."
          - name: SYN_OPTIC_HTTPS_PORT
            value: "4443"
          - name: SYN_OPTIC_AXON
            value: "aha://axon..."
```

(continues on next page)

(continued from previous page)

```
          - name: SYN_OPTIC_CORTEX
            value: "aha://cortex..."
          - name: SYN_OPTIC_JSONSTOR
            value: "aha://jsonstor..."
        volumeMounts:
          - mountPath: /vertex/storage
            name: data
        imagePullPolicy: Always
        startupProbe:
          failureThreshold: 2147483647
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
        readinessProbe:
          failureThreshold: 2
          initialDelaySeconds: 20
          timeoutSeconds: 20
          periodSeconds: 20
          exec:
            command: ['python', '-m', 'synapse.tools.healthcheck', '-c', 'cell:///vertex/
↪storage']
      restartPolicy: Always
      imagePullSecrets:
        - name: "regcred"
---
apiVersion: v1
kind: Service
metadata:
  name: optic
  labels:
    app.kubernetes.io/name: "optic"
    app.kubernetes.io/instance: "optic00"
    app.kubernetes.io/version: "v2.x.x"
    app.kubernetes.io/component: "optic"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  type: ClusterIP
  selector:
    app.kubernetes.io/name: optic
    environment: "dev"
  ports:
    - port: 4443
      protocol: TCP
      name: https
```

Before we deploy that, we need to create the Aha provisioning URL. We do this via `kubectl exec`. That should look like the following:

```
$ kubectl exec deployment/aha -- python -m synapse.tools.aha.provision.service 00.optic
```

(continues on next page)

(continued from previous page)

```
one-time use URL: ssl://aha.default.svc.cluster.local:27272/
↪3f692cda9dfb152f74a8a0251165bcc4?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f
```

We want to copy that URL into the SYN_OPTIC_AHA_PROVISION environment variable, so that block looks like the following:

```
- name: SYN_OPTIC_AHA_PROVISION
  value: "ssl://aha.default.svc.cluster.local:27272/3f692cda9dfb152f74a8a0251165bcc4?
↪certhash=09c8329ed29b89b77e0a2fdc23e64aea407ad4d7e71d67d3fea92ddd9466592f"
```

This can then be deployed via `kubectl apply`:

```
$ kubectl apply -f optic.yaml
persistentvolumeclaim/example-optic00 created
deployment.apps/optic00 created
service/optic created
```

You can see the Optic logs as well. These show provisioning and listening for traffic, as well as the connection being made to the Axon, Cortex, and JSONStor services:

```
$ kubectl logs --tail 30 -l app.kubernetes.io/instance=optic00
2023-03-08 17:32:40,149 [INFO] log level set to DEBUG [common.
↪py:setlogging:MainThread:MainProcess]
2023-03-08 17:32:40,150 [DEBUG] Set config valu from envar: [SYN_OPTIC_CORTEX] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,150 [DEBUG] Set config valu from envar: [SYN_OPTIC_AXON] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,151 [DEBUG] Set config valu from envar: [SYN_OPTIC_JSONSTOR] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,151 [DEBUG] Set config valu from envar: [SYN_OPTIC_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,152 [DEBUG] Set config valu from envar: [SYN_OPTIC_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,153 [INFO] Provisioning optic from AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:32:40,264 [DEBUG] Set config valu from envar: [SYN_OPTIC_CORTEX] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,265 [DEBUG] Set config valu from envar: [SYN_OPTIC_AXON] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,265 [DEBUG] Set config valu from envar: [SYN_OPTIC_JSONSTOR] [config.
↪py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,265 [DEBUG] Set config valu from envar: [SYN_OPTIC_HTTPS_PORT]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:40,266 [DEBUG] Set config valu from envar: [SYN_OPTIC_AHA_PROVISION]␣
↪[config.py:setConfFromEnvs:MainThread:MainProcess]
2023-03-08 17:32:45,181 [INFO] Done provisioning optic AHA service. [cell.py:_
↪bootCellProv:MainThread:MainProcess]
2023-03-08 17:32:45,247 [INFO] optic wwwroot: /usr/local/lib/python3.10/dist-packages/
↪optic/site [app.py:initServiceStorage:MainThread:MainProcess]
2023-03-08 17:32:45,248 [WARNING] Waiting for remote jsonstor... [app.
↪py:initJsonStor:MainThread:MainProcess]
2023-03-08 17:32:45,502 [INFO] Connected to JsonStor at [aha://jsonstor...] [app.
```

(continues on next page)

(continued from previous page)

```
↪py:initJsonStor:MainThread:MainProcess]
2023-03-08 17:32:45,504 [INFO] Waiting for connection to Cortex [app.py:_
↪initOpticCortex:MainThread:MainProcess]
2023-03-08 17:32:45,599 [INFO] Connected to Cortex at [aha://cortex...] [app.py:_
↪initOpticCortex:MainThread:MainProcess]
2023-03-08 17:32:45,930 [INFO] Connected to Axon at [aha://axon...] [app.
↪py:onaxonlink:MainThread:MainProcess]
2023-03-08 17:32:45,937 [DEBUG] Email settings/server not configured or invalid. [app.
↪py:initEmailApis:asyncio_0:MainProcess]
2023-03-08 17:32:45,975 [INFO] dmon listening: ssl://0.0.0.0:0?hostname=00.optic.default.
↪svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
2023-03-08 17:32:45,976 [WARNING] NO CERTIFICATE FOUND! generating self-signed␣
↪certificate. [cell.py:addHttpsPort:MainThread:MainProcess]
2023-03-08 17:32:47,773 [INFO] https listening: 4443 [cell.
↪py:initServiceNetwork:MainThread:MainProcess]
2023-03-08 17:32:47,773 [INFO] ...optic API (telepath): ssl://0.0.0.0:0?hostname=00.
↪optic.default.svc.cluster.local&ca=default.svc.cluster.local [cell.
↪py:initFromArgv:MainThread:MainProcess]
2023-03-08 17:32:47,773 [INFO] ...optic API (https): 4443 [cell.
↪py:initFromArgv:MainThread:MainProcess]
```

Once Optic is connected, we will need to set a password for the user we previously created in order to log in. This can be done via `kubectl exec`, setting the password for the user on the Cortex:

```
$ kubectl exec -it deployment/cortex00 -- python -m synapse.tools.moduser --passwd␣
↪secretPassword visi
Modifying user: visi
...setting passwd: secretPassword
```

Enable a port-forward to connect to the Optic service:

```
$ kubectl port-forward service/optic 4443:https
```

You can then use a Chrome browser to navigate to `https://localhost:4443` and you should be prompted with an Optic login screen. You can enter your username and password (`visi` and `secretPassword`) in order to login to Optic.

Practical Considerations

The following items should be considered for Kubernetes deployments intended for production use cases:

Healthchecks
: These examples use large `startupProbe` failure values. Vertex recommends these large values, since service updates may have automatic data migrations which they perform at startup. These will be performed before a service has enabled any listeners which would respond to healthcheck probes. The large value prevents a service from being terminated prior to a long running data migration completing.

Ingress and Load Balancing
: The use of `kubectl port-forward` may not be sustainable in a production environment. It is common to use a form of ingress controller or load balancer for external services to reach services such as the Cortex or Optic applications. It is common for the Optic UI or the Cortex HTTP API to be

exposed to end users since that often has a simpler networking configuration than exposing Telepath services on Aha and the Cortex.

Log aggregation
Many Kubernetes clusters may perform some sort of log aggregation for the containers running in them. If your log aggregation solution can parse JSON formatted container logs, you can set the `SYN_LOG_STRUCT` environment variable to `"true"` to enable structured log output. See *Configure Logging* for more information about that option.

Node Selectors
These examples do not use any node selectors to bind pods to specific nodes or node types. Node selectors on the podspec can be used to constrain different services to different types of nodes. For example, they can be used to ensure the Cortex is deployed to a node which has been provisioned as a high memory node for that purpose.

PVC
The previous examples used relatively small volume claim sizes for demonstration purposes. A `storageClass` which can be dynamically resized will be helpful in the event of needing to grow the storage used by a deployment. This is a common feature for managed Kubernetes instances.

Performance Tuning in Kubernetes

It is common for Kubernetes to be executed in a managed environment, where an operator may not have direct access to the underlying hosts. In that scenario, applying the system configurations detailed in *Performance Tuning* may be difficult. The following example shows a DaemonSet which runs a privileged pod, that ensures that the desired `sysctl` values are set on the host. You may need to modify this to meet any requirements which are specific to your deployment.

The following `sysctl.yaml` can be used as the basis to deploy these modifications.

```yaml
apiVersion: "apps/v1"
kind: "DaemonSet"
metadata:
  name: "setsysctl"
  labels:
    app.kubernetes.io/name: "sysctl"
    app.kubernetes.io/instance: "sysctl"
    app.kubernetes.io/version: "1.36.0-glibc"
    app.kubernetes.io/component: "sysctl"
    app.kubernetes.io/part-of: "synapse"
    environment: "dev"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "sysctl"
      app.kubernetes.io/instance: "sysctl"
      app.kubernetes.io/version: "1.36.0-glibc"
      app.kubernetes.io/component: "sysctl"
      app.kubernetes.io/part-of: "synapse"
      environment: "dev"
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "sysctl"
        app.kubernetes.io/instance: "sysctl"
```

(continues on next page)

(continued from previous page)

```
        app.kubernetes.io/version: "1.36.0-glibc"
        app.kubernetes.io/component: "sysctl"
        app.kubernetes.io/part-of: "synapse"
        environment: "dev"
    spec:
      containers:
      - name: "apply-sysctl"
        image: "busybox:1.36.0-glibc"  # Latest glibc based busybox
        securityContext:
          privileged: true
        command:
        - "/bin/sh"
        - "-c"
        - |
          set -o errexit
          set -o xtrace
          while sysctl -w vm.swappiness=10 vm.dirty_expire_centisecs=20 vm.dirty_
↪writeback_centisecs=20 vm.dirty_background_ratio=2 vm.dirty_ratio=4
          do
            sleep 600s
          done
```

This can be deployed via `kubectl apply`. That will create the DaemonSet for you..

```
$ kubectl apply -f sysctl_dset.yaml
daemonset.apps/setsysctl created
```

You can see the sysctl pods by running the following command:

```
$ kubectl get pods -l app.kubernetes.io/component=sysctl -o wide
```

6.4.2 AHA Configuration Options

aha:admin

An AHA client certificate CN to register as a local admin user.

Type
    `string`

Environment Variable
    `SYN_AHA_AHA_ADMIN`

aha:leader

The AHA service name to claim as the active instance of a storm service.

Type
`string`

Environment Variable
`SYN_AHA_AHA_LEADER`

aha:name

The name of the cell service in the aha service registry.

Type
`string`

Environment Variable
`SYN_AHA_AHA_NAME`

aha:network

The AHA service network. This makes aha:name/aha:leader relative names.

Type
`string`

Environment Variable
`SYN_AHA_AHA_NETWORK`

aha:provision

The telepath URL of the aha provisioning service.

Type
`['string', 'array']`

Environment Variable
`SYN_AHA_AHA_PROVISION`

aha:registry

The telepath URL of the aha service registry.

Type
`['string', 'array']`

Environment Variable
`SYN_AHA_AHA_REGISTRY`

aha:urls

A list of all available AHA server URLs.

Type
    ['string', 'array']

Environment Variable
    SYN_AHA_AHA_URLS

aha:user

The username of this service when connecting to others.

Type
    string

Environment Variable
    SYN_AHA_AHA_USER

auth:anon

Allow anonymous telepath access by mapping to the given user name.

Type
    string

Environment Variable
    SYN_AHA_AUTH_ANON

auth:passwd

Set to <passwd> (local only) to bootstrap the root user password.

Type
    string

Environment Variable
    SYN_AHA_AUTH_PASSWD

backup:dir

A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.

Type
    string

Environment Variable
    SYN_AHA_BACKUP_DIR

dmon:listen

A config-driven way to specify the telepath bind URL.

Type
`['string', 'null']`

Environment Variable
`SYN_AHA_DMON_LISTEN`

https:headers

Headers to add to all HTTPS server responses.

Type
`object`

Environment Variable
`SYN_AHA_HTTPS_HEADERS`

https:parse:proxy:remoteip

Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_AHA_HTTPS_PARSE_PROXY_REMOTEIP`

https:port

A config-driven way to specify the HTTPS port.

Type
`['integer', 'null']`

Environment Variable
`SYN_AHA_HTTPS_PORT`

limit:disk:free

Minimum disk free space percentage before setting the cell read-only.

Type
`['integer', 'null']`

Default Value
`5`

Environment Variable
`SYN_AHA_LIMIT_DISK_FREE`

max:users

Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).

Type
`integer`

Default Value
`0`

Environment Variable
`SYN_AHA_MAX_USERS`

mirror

A telepath URL for our upstream mirror (we must be a backup!).

Type
`['string', 'null']`

Environment Variable
`SYN_AHA_MIRROR`

nexslog:en

Record all changes to a stream file on disk. Required for mirroring (on both sides).

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_AHA_NEXSLOG_EN`

onboot:optimize

Delay startup to optimize LMDB databases during boot to recover free space and increase performance. This may take a while.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_AHA_ONBOOT_OPTIMIZE`

provision:listen

A telepath URL for the AHA provisioning listener.

Type
    ['string', 'null']

Environment Variable
    SYN_AHA_PROVISION_LISTEN

6.4.3 Axon Configuration Options

aha:admin

An AHA client certificate CN to register as a local admin user.

Type
    string

Environment Variable
    SYN_AXON_AHA_ADMIN

aha:leader

The AHA service name to claim as the active instance of a storm service.

Type
    string

Environment Variable
    SYN_AXON_AHA_LEADER

aha:name

The name of the cell service in the aha service registry.

Type
    string

Environment Variable
    SYN_AXON_AHA_NAME

aha:network

The AHA service network. This makes aha:name/aha:leader relative names.

Type
    string

Environment Variable
    SYN_AXON_AHA_NETWORK

aha:provision

The telepath URL of the aha provisioning service.

Type
`['string', 'array']`

Environment Variable
`SYN_AXON_AHA_PROVISION`

aha:registry

The telepath URL of the aha service registry.

Type
`['string', 'array']`

Environment Variable
`SYN_AXON_AHA_REGISTRY`

aha:user

The username of this service when connecting to others.

Type
`string`

Environment Variable
`SYN_AXON_AHA_USER`

auth:anon

Allow anonymous telepath access by mapping to the given user name.

Type
`string`

Environment Variable
`SYN_AXON_AUTH_ANON`

auth:passwd

Set to <passwd> (local only) to bootstrap the root user password.

Type
`string`

Environment Variable
`SYN_AXON_AUTH_PASSWD`

backup:dir

A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.

Type
    string

Environment Variable
    SYN_AXON_BACKUP_DIR

dmon:listen

A config-driven way to specify the telepath bind URL.

Type
    ['string', 'null']

Environment Variable
    SYN_AXON_DMON_LISTEN

http:proxy

An aiohttp-socks compatible proxy URL to use in the wget API.

Type
    string

Environment Variable
    SYN_AXON_HTTP_PROXY

https:headers

Headers to add to all HTTPS server responses.

Type
    object

Environment Variable
    SYN_AXON_HTTPS_HEADERS

https:parse:proxy:remoteip

Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.

Type
    boolean

Default Value
    False

Environment Variable
    SYN_AXON_HTTPS_PARSE_PROXY_REMOTEIP

https:port

A config-driven way to specify the HTTPS port.

Type
`['integer', 'null']`

Environment Variable
`SYN_AXON_HTTPS_PORT`

limit:disk:free

Minimum disk free space percentage before setting the cell read-only.

Type
`['integer', 'null']`

Default Value
`5`

Environment Variable
`SYN_AXON_LIMIT_DISK_FREE`

max:bytes

The maximum number of bytes that can be stored in the Axon.

Type
`integer`

Environment Variable
`SYN_AXON_MAX_BYTES`

max:count

The maximum number of files that can be stored in the Axon.

Type
`integer`

Environment Variable
`SYN_AXON_MAX_COUNT`

max:users

Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).

Type
`integer`

Default Value
`0`

Environment Variable
`SYN_AXON_MAX_USERS`

nexslog:en

Record all changes to a stream file on disk. Required for mirroring (on both sides).

Type
  `boolean`

Default Value
  `False`

Environment Variable
  `SYN_AXON_NEXSLOG_EN`

onboot:optimize

Delay startup to optimize LMDB databases during boot to recover free space and increase performance. This may take a while.

Type
  `boolean`

Default Value
  `False`

Environment Variable
  `SYN_AXON_ONBOOT_OPTIMIZE`

tls:ca:dir

An optional directory of CAs which are added to the TLS CA chain for wget and wput APIs.

Type
  `string`

Environment Variable
  `SYN_AXON_TLS_CA_DIR`

6.4.4 JSONStor Configuration Options

aha:admin

An AHA client certificate CN to register as a local admin user.

Type
  `string`

Environment Variable
  `SYN_JSONSTOR_AHA_ADMIN`

aha:leader

The AHA service name to claim as the active instance of a storm service.

Type
string

Environment Variable
SYN_JSONSTOR_AHA_LEADER

aha:name

The name of the cell service in the aha service registry.

Type
string

Environment Variable
SYN_JSONSTOR_AHA_NAME

aha:network

The AHA service network. This makes aha:name/aha:leader relative names.

Type
string

Environment Variable
SYN_JSONSTOR_AHA_NETWORK

aha:provision

The telepath URL of the aha provisioning service.

Type
['string', 'array']

Environment Variable
SYN_JSONSTOR_AHA_PROVISION

aha:registry

The telepath URL of the aha service registry.

Type
['string', 'array']

Environment Variable
SYN_JSONSTOR_AHA_REGISTRY

aha:user

The username of this service when connecting to others.

Type
    string

Environment Variable
    SYN_JSONSTOR_AHA_USER

auth:anon

Allow anonymous telepath access by mapping to the given user name.

Type
    string

Environment Variable
    SYN_JSONSTOR_AUTH_ANON

auth:passwd

Set to <passwd> (local only) to bootstrap the root user password.

Type
    string

Environment Variable
    SYN_JSONSTOR_AUTH_PASSWD

backup:dir

A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.

Type
    string

Environment Variable
    SYN_JSONSTOR_BACKUP_DIR

dmon:listen

A config-driven way to specify the telepath bind URL.

Type
    ['string', 'null']

Environment Variable
    SYN_JSONSTOR_DMON_LISTEN

https:headers

Headers to add to all HTTPS server responses.

Type
`object`

Environment Variable
`SYN_JSONSTOR_HTTPS_HEADERS`

https:parse:proxy:remoteip

Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_JSONSTOR_HTTPS_PARSE_PROXY_REMOTEIP`

https:port

A config-driven way to specify the HTTPS port.

Type
`['integer', 'null']`

Environment Variable
`SYN_JSONSTOR_HTTPS_PORT`

limit:disk:free

Minimum disk free space percentage before setting the cell read-only.

Type
`['integer', 'null']`

Default Value
`5`

Environment Variable
`SYN_JSONSTOR_LIMIT_DISK_FREE`

max:users

Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).

Type
`integer`

Default Value
`0`

Environment Variable
`SYN_JSONSTOR_MAX_USERS`

nexslog:en

Record all changes to a stream file on disk. Required for mirroring (on both sides).

Type
  `boolean`

Default Value
  `False`

Environment Variable
  `SYN_JSONSTOR_NEXSLOG_EN`

onboot:optimize

Delay startup to optimize LMDB databases during boot to recover free space and increase performance. This may take a while.

Type
  `boolean`

Default Value
  `False`

Environment Variable
  `SYN_JSONSTOR_ONBOOT_OPTIMIZE`

6.4.5 Cortex Configuration Options

aha:admin

An AHA client certificate CN to register as a local admin user.

Type
  `string`

Environment Variable
  `SYN_CORTEX_AHA_ADMIN`

aha:leader

The AHA service name to claim as the active instance of a storm service.

Type
  `string`

Environment Variable
  `SYN_CORTEX_AHA_LEADER`

aha:name

The name of the cell service in the aha service registry.

Type
    `string`

Environment Variable
    `SYN_CORTEX_AHA_NAME`

aha:network

The AHA service network. This makes aha:name/aha:leader relative names.

Type
    `string`

Environment Variable
    `SYN_CORTEX_AHA_NETWORK`

aha:provision

The telepath URL of the aha provisioning service.

Type
    `['string', 'array']`

Environment Variable
    `SYN_CORTEX_AHA_PROVISION`

aha:registry

The telepath URL of the aha service registry.

Type
    `['string', 'array']`

Environment Variable
    `SYN_CORTEX_AHA_REGISTRY`

aha:user

The username of this service when connecting to others.

Type
    `string`

Environment Variable
    `SYN_CORTEX_AHA_USER`

auth:anon

Allow anonymous telepath access by mapping to the given user name.

Type
    string

Environment Variable
    SYN_CORTEX_AUTH_ANON

auth:passwd

Set to <passwd> (local only) to bootstrap the root user password.

Type
    string

Environment Variable
    SYN_CORTEX_AUTH_PASSWD

axon

A telepath URL for a remote axon.

Type
    string

Environment Variable
    SYN_CORTEX_AXON

backup:dir

A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.

Type
    string

Environment Variable
    SYN_CORTEX_BACKUP_DIR

cron:enable

Deprecated. This option no longer controls cron execution and will be removed in Synapse 3.0.

Type
    boolean

Default Value
    True

Environment Variable
    SYN_CORTEX_CRON_ENABLE

dmon:listen

A config-driven way to specify the telepath bind URL.

Type
`['string', 'null']`

Environment Variable
`SYN_CORTEX_DMON_LISTEN`

http:proxy

An aiohttp-socks compatible proxy URL to use storm HTTP API.

Type
`string`

Environment Variable
`SYN_CORTEX_HTTP_PROXY`

https:headers

Headers to add to all HTTPS server responses.

Type
`object`

Environment Variable
`SYN_CORTEX_HTTPS_HEADERS`

https:parse:proxy:remoteip

Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_CORTEX_HTTPS_PARSE_PROXY_REMOTEIP`

https:port

A config-driven way to specify the HTTPS port.

Type
`['integer', 'null']`

Environment Variable
`SYN_CORTEX_HTTPS_PORT`

jsonstor

A telepath URL for a remote jsonstor.

Type
`string`

Environment Variable
`SYN_CORTEX_JSONSTOR`

layer:lmdb:map_async

Set the default lmdb:map_async value in LMDB layers.

Type
`boolean`

Default Value
`True`

Environment Variable
`SYN_CORTEX_LAYER_LMDB_MAP_ASYNC`

layer:lmdb:max_replay_log

Set the max size of the replay log for all layers.

Type
`integer`

Default Value
`10000`

Environment Variable
`SYN_CORTEX_LAYER_LMDB_MAX_REPLAY_LOG`

layers:lockmemory

Should new layers lock memory for performance by default.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_CORTEX_LAYERS_LOCKMEMORY`

layers:logedits

Whether nodeedits are logged in each layer.

Type
`boolean`

Default Value
`True`

Environment Variable
`SYN_CORTEX_LAYERS_LOGEDITS`

limit:disk:free

Minimum disk free space percentage before setting the cell read-only.

Type
`['integer', 'null']`

Default Value
`5`

Environment Variable
`SYN_CORTEX_LIMIT_DISK_FREE`

max:nodes

Maximum number of nodes which are allowed to be stored in a Cortex.

Type
`integer`

Environment Variable
`SYN_CORTEX_MAX_NODES`

max:users

Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).

Type
`integer`

Default Value
`0`

Environment Variable
`SYN_CORTEX_MAX_USERS`

mirror

A telepath URL for our upstream mirror (we must be a backup!).

Type
`['string', 'null']`

Environment Variable
`SYN_CORTEX_MIRROR`

modules

A list of module classes to load.

Type
`array`

Default Value
`[]`

Environment Variable
`SYN_CORTEX_MODULES`

nexslog:en

Record all changes to a stream file on disk. Required for mirroring (on both sides).

Type
`boolean`

Default Value
`True`

Environment Variable
`SYN_CORTEX_NEXSLOG_EN`

onboot:optimize

Delay startup to optimize LMDB databases during boot to recover free space and increase performance. This may take a while.

Type
`boolean`

Default Value
`False`

Environment Variable
`SYN_CORTEX_ONBOOT_OPTIMIZE`

storm:interface:scrape

Enable Storm scrape interfaces when using $lib.scrape APIs.

Type
    `boolean`

Default Value
    `True`

Environment Variable
    `SYN_CORTEX_STORM_INTERFACE_SCRAPE`

storm:interface:search

Enable Storm search interfaces for lookup mode.

Type
    `boolean`

Default Value
    `True`

Environment Variable
    `SYN_CORTEX_STORM_INTERFACE_SEARCH`

storm:log

Log storm queries via system logger.

Type
    `boolean`

Default Value
    `False`

Environment Variable
    `SYN_CORTEX_STORM_LOG`

storm:log:level

Logging log level to emit storm logs at.

Type
    `['integer', 'string']`

Default Value
    `'INFO'`

Environment Variable
    `SYN_CORTEX_STORM_LOG_LEVEL`

tls:ca:dir

An optional directory of CAs which are added to the TLS CA chain for Storm HTTP API calls.

Type
    string

Environment Variable
    SYN_CORTEX_TLS_CA_DIR

trigger:enable

Deprecated. This option no longer controls trigger execution and will be removed in Synapse 3.0.

Type
    boolean

Default Value
    True

Environment Variable
    SYN_CORTEX_TRIGGER_ENABLE

CHAPTER

SEVEN

SYNAPSE DEVELOPER GUIDE

This Dev Guide is written by and for Synapse developers.

Note: Synapse as a library is under constant development. It is posssible that content here may become out of date. If you encounter issues with documentation in the Developers guides, please reach out to us on our Synapse Slack chat or file an issue in our projects Github page.

The Dev Guide is a living document and will continue to be updated and expanded as appropriate. The current sections are:

7.1 Rapid Power-Up Development

Developing Rapid Power-Ups allows Synapse power users to extend the capabilities of the Storm query language, provides ways to implement use-case specific commands, embed documentation, and even implement customized visual workflows in **Optic**, the commercial Synapse UI.

A Rapid Power-Up consists of a **Storm Package** which is a JSON object which defines everything used to extend the Storm language and provide additional documentation. **Storm Packages** can be loaded directly into your **Cortex**.

In this guide we will discuss the basics of **Storm Package** development and discuss a few best practices you can use to ensure they are secure, powerful, and easy to use.

The example `acme-hello` power-up discussed in this guide is included in the **Synapse** repository within the `examples/power-ups/rapid/acme-hello` folder. You can find that at Acme-Hello Example.

7.1.1 Anatomy of a Storm Package

A **Storm Package** consists of a YAML file which defines the various commands, modules, documentation, and workflows embedded within the package.

Minimal Example

As you can see in the minimal example below, the **Storm Package** is defined by a YAML file that gets processed and loaded into your Cortex.

acme-hello.yaml:

```
name: acme-hello
version: 0.0.1

synapse_version: '>=2.144.0,<3.0.0'

genopts:
    dotstorm: true # Specify that storm command/module files end with ".storm"

author:
  url: https://acme.newp
  name: ACME Explosives and Anvils

desc: Acme-Hello is a minimal example of a Rapid Power-Up.

modules:
  - name: acme.hello
  - name: acme.hello.privsep
    asroot:perms:
        - [ acme, hello, user ]

commands:
  - name: acme.hello.sayhi
    descr: Print the hello message.
```

Note: First, a note on namespacing. To ensure your **Storm Package** is going to play well with other packages, it is important to choose an appropriate namespace for your power-up. In this case, the acme part of the name is meant to be replaced with your company name or an abbreviated version of it. The hello part is meant to be replaced with an indicator of the type of functionality the **Storm Package** contains.

Namespace now, thank yourself later.

When you define commands and modules, they will be loaded from files using the location of the **Storm Package** YAML file to locate their contents:

```
acme-hello.yaml
storm/

    modules/
        acme.hello.storm
        acme.hello.privsep.storm

    commands/
        acme.hello.sayhi.storm
```

storm/modules/acme.hello.storm:

```
function woot(text) {
    $lib.print($text)
    return($lib.null)
}
```

storm/commands/acme.hello.sayhi.storm:

```
$hello = $lib.import(acme.hello)
$hello.woot("hello storm!")
```

Building / Loading

To build and load **Storm Packages**, use the genpkg tool included within Synapse. For this example, we will assume you have deployed your Synapse environment according to the Deployment Guide:

```
python -m synapse.tools.genpkg acme-hello.yaml --push aha://cortex...
```

Note: If you added an alternate admin user or used a non-standard naming convention you may need to adjust the aha://cortex... telepath URL to connect to your Cortex.

Once your **Storm Package** has loaded successfully, you can use the **Storm** CLI to see it in action:

```
invisigoth@visi01:~$ python -m synapse.tools.storm aha://cortex...

Welcome to the Storm interpreter!

Local interpreter (non-storm) commands may be executed with a ! prefix:
    Use !quit to exit.
    Use !help to see local interpreter commands.

storm> acme.hello.sayhi
hello storm!
complete. 0 nodes in 1 ms (0/sec).
storm>
```

7.1.2 Storm Modules

Deploying **Storm Modules** allows you to author powerful library functions that you can use in automation or **Storm Commands** to facilitate code re-use and enforce privilege separation boundaries.

A **Storm Module** is specified within the modules: section of the **Storm Package** YAML file.

```
modules:

  - name: acme.hello
    modconf:
        varname: varvalu
        othervar: [1, 2, 3]
```

The modconf: key can be used to specify variables which will be mapped into the module's **Storm** runtime and accessible using the implicit variable $modconf:

```
function foo() {
    $lib.print($modconf.varname)
    return((10))
}

function bar() {
    for $i in $modconf.othervar {
        // Do something using $i...
    }
}
```

Privileged Modules

In order to facilitate delegating permission for privileged operations, **Storm** modules may specify permissions which allow the module to be imported with admin privileges. It is a best-practice to declare these permissions within the **Storm** package using the `perms:` key before using them:

```
perms:
  - perm: [ acme, hello, user ]
    gate: cortex
    desc: Allows a user to call privileged APIs from Acme-Hello.

modules:

  - name: acme.hello.privsep
    asroot:perms:
        - [ acme, hello, user ]
```

To minimize risk, you must very carefully consider what functions to implement within a privileged **Storm** module! Privileged modules should contain the absolute minimum required functionality.

An excellent example use case for a privileged **Storm** module exists when you have an API key or password which you would like to use on a user's behalf without disclosing the actual API key. The **Storm** library `$lib.globals.set(<name>, <valu>)` and `$lib.globals.get(<name>)` can be used to access protected global variables which regular users may not access without special permissions. By implementing a privileged **Storm** module which retrieves the API key and uses it on the user's behalf without disclosing it, you may protect the API key from disclosure while also allowing users to use it. For example, `acme.hello.privsep.storm`:

```
function getFooByBar(bar) {

    // Retrieve an API key from protected storage
    $apikey = $lib.globals.get(acme:hello:apikey)

    $headers = ({
        "apikey": $apikey
    })

    $url = $lib.str.format("https://acme.newp/api/v1/foo/{bar}", bar=$bar)

    // Use the API key on the callers behalf
    $resp = $lib.inet.http.get($url, headers=$headers)
    if ($resp.code != 200) {
```

(continues on next page)

(continued from previous page)

```
        $lib.warn("/api/v1/foo returned HTTP code: {code}", code=$resp.code)
        return($lib.null)
    }

    // Return the JSON response (but not the API key)
    return($resp.json())
}
```

Notice that the `$apikey` is being retrieved and used to call the HTTP API but is not returned to the caller.

7.1.3 Storm Commands

Adding **Storm Commands** to your Cortex via a **Storm Package** is a great way to extend the functionality of your Cortex in a CLI user-friendly way.

Command Line Options

Every **Storm** command has the `--help` option added automatically. This means that it is always safe to execute any command with `--help` to get a usage statement and enumerate command line arguments. The `desc` field specified in the command is included in the output:

```
storm> acme.hello.sayhi --help

Print the hello message.

Usage: acme.hello.sayhi [options]

Options:

  --help                      : Display the command usage.
complete. 0 nodes in 4 ms (0/sec).
storm>
```

Storm Commands may specify command line arguments using a convention which is similar (although not identical to) Python's `argparse` library.

A more complex command declaration:

```
commands:

  - name: acme.hello.omgopts
    descr: |
        This is a multi-line description containing usage examples.

        // Run the command with some nodes
        inet:fqdn=acme.newp | acme.hello.omgopts vertex.link

        // Run the command with some command line switches
        acme.hello.omgopts --debug --hehe haha vertex.link

    cmdargs:
```

(continues on next page)

(continued from previous page)

```
    - - --hehe
      - type: str
        help: The value of the hehe optional input.

    - - --debug
      - type: bool
        default: false
        action: store_true
        help: Enable debug output.

    - - fqdn
      - type: str
        help: A mandatory / positional command line argument.
```

A more complete example of help output:

```
storm> acme.hello.omgopts --help

This is a multi-line description containing usage examples.

// Run the command with some nodes
inet:fqdn=acme.newp | acme.hello.omgopts vertex.link

// Run the command with some command line switches
acme.hello.omgopts --debug --hehe haha vertex.link


Usage: acme.hello.omgopts [options] <fqdn>

Options:

  --help                      : Display the command usage.
  --hehe <hehe>               : The value of the hehe optional input.
  --debug                     : Enable debug output.

Arguments:

  <fqdn>                      : A mandatory / positional command line argument.
complete. 0 nodes in 6 ms (0/sec).
```

Command line options are available within the **Storm** command by accessing the implicit `$cmdopts` variable.

`storm/commands/acme.hello.omgopts.storm`:

```
// An init {} block only runs once even if there are multiple nodes in the pipeline.

init {

    // Set global debug (once) if the user specified --debug
    if $cmdopts.debug { $lib.debug = $lib.true }

    if ($cmdopts.hehe) { $lib.print("User Specified hehe: {hehe}", hehe=$cmdopts.hehe) }
```

(continues on next page)

(continued from previous page)

```
    // Normalize the FQDN in case we want to send it to an external system
    ($ok, $fqdn) = $lib.trycast(inet:fqdn, $cmdopts.fqdn)
    if (not $ok) {
        $lib.exit("Invalid FQDN Specified: {fqdn}", fqdn=$cmdopts.fqdn)
    }

    // Maybe call an API here or something...
    $lib.print("FQDN: {fqdn}", fqdn=$fqdn)
}


// You may also act on nodes in the pipeline
$lib.print("GOT NODE: {repr}", repr=$node.repr())

if $lib.debug { $lib.print("debug mode detected!") }

// Any nodes still in the pipeline are sent as output
```

Command Option Conventions

--help
: This option is reserved and handled automatically to print a command usage statement which also enumerates any positional or optional arguments.

--debug
: This option is typically used to enable debug output in the **Storm** interpreter by setting the `$lib.debug` variable if it is specified. The `$lib.debug` variable has a recursive effect and will subsequently enable debug output in any command or functions called from the command.

--yield
: By default, a command is generally expected to yield the nodes that it received as input from the pipeline. In some instances it is useful to instruct the command to yield the nodes it creates. For example, if you specify `inet:fqdn` nodes as input to a DNS resolver command, it may be useful to tell the command to yield the newly created `inet:dns:a` records rather than the input `inet:fqdn` nodes. Commands frequently use the `divert` **Storm** command to implement `--yield` functionality.

--asof <time>
: To minimize duplicate API calls, many **Storm** packages cache results using the `$lib.jsonstor` API. When caching is in use, the `--asof <time>` option is used to control cache aging. Users may specify `--asof now` to disable caching.

7.1.4 Specifying Documentation

Documentation may be specified in the **Storm Package** file that will embed `markdown` documentation into the package. While there are not currently any CLI tools to view/use this documentation, it is presented in the **Power-Ups** tab in the **Help Tool** within the commercial *Synapse User Interface*.

Markdown documents may be specified for inclusion by adding a `docs:` section to the **Storm Package** YAML file:

```
docs:
    - title: User Guide
      path: docs/userguide.md
```

(continues on next page)

(continued from previous page)

```
  - title: Admin Guide
    path: docs/adminguide.md
  - title: Changelog
    path: docs/changelog.md
```

7.1.5 Testing Storm Packages

It is **highly** recommended that any production **Storm Packages** use development "best practices" including version control and unit testing. For the `acme-hello` example, we have included a test that you can use as an example to expand on.

`test_acme_hello.py`:

```
import os

import synapse.tests.utils as s_test

dirname = os.path.abspath(os.path.dirname(__file__))

class AcmeHelloTest(s_test.StormPkgTest):

    assetdir = os.path.join(dirname, 'testassets')
    pkgprotos = (os.path.join(dirname, 'acme-hello.yaml'),)

    async def test_acme_hello(self):

        async with self.getTestCore() as core:

            msgs = await core.stormlist('acme.hello.sayhi')
            self.stormIsInPrint('hello storm!', msgs)
            self.stormHasNoWarnErr(msgs)
```

With the file `test_acme_hello.py` located in the same directory as `acme-hello.yaml` you can use the standard `pytest` invocation to run the test:

```
python -m pytest -svx test_acme_hello.py
```

7.1.6 Advanced Features

Using `divert` to implement `--yield`

The `--yield` option is typically used to allow a **Storm** command which takes nodes as input to optionally output the new nodes it added rather than the nodes it received as input. The `divert` command was added to **Storm** to simplify implementing this convention.

To implement a command with a `--yield` option is typically accomplished via the following pattern:

```
commands:

  - name: acme.hello.mayyield
    descr: |
```

(continues on next page)

(continued from previous page)

```
        Take in an FQDN and make DNS A records to demo --yield

        inet:fqdn=vertex.link | acme.hello.mayyield

    cmdargs:

      - - --yield
        - default: false
          action: store_true
          help: Yield the newly created inet:dns:a records rather than the input␣
↪inet:fqdn nodes.
```

Then within storm/commands/acme.hello.mayyield.storm:

```
function nodeGenrFunc(fqdn) {
    // Fake a DNS lookup and make a few inet:dns:a records...
    [ inet:dns:a=($fqdn, 1.2.3.4) ]
    [ inet:dns:a=($fqdn, 123.123.123.123) ]
}

divert --yield $cmdopts.yield $nodeGenrFunc($node)
```

When executed, the `acme.hello.mayyield` command will output the nodes received as inputs which is useful for pipelining enrichments. If the user specifies `--yield` the command will output the resulting `inet:dns:a` nodes constructed by the `nodeGenrFunc()` function.

Optic Actions

If you have access to the **Synapse** commercial UI **Optic** you may find it helpful to embed **Optic** actions within your **Storm Package**. These actions will be presented to users in the context-menu when they right-click on nodes within **Optic**.

To define **Optic** actions, you declare them in the **Storm Package** YAML file:

```
optic:
    actions:
      - name: Hello Omgopts
        storm: acme.hello.omgopts --debug
        descr: This description is displayed as the tooltip in the menu
        forms: [ inet:ipv4, inet:fqdn ]
```

By specifying the `forms:` key, you can control which node actions will be presented on different forms. For example, if you are writing a DNS power-up, you may want to limit the specified actions to `inet:ipv4`, `inet:ipv6`, and `inet:fqdn` nodes.

When selected, the query specified in the `storm:` key will be run with the currently selected nodes as input. For example, if you right-click on the node `inet:fqdn=vertex.link` and select `actions -> acme-hello -> Hello Omgopts` it will execute the specified query as though it were run like this:

```
inet:fqdn=vertex.link | acme.hello.omgopts --debug
```

Any printed output, including warnings, will be displayed in the **Optic** `Console Tool`.

7.2 Synapse Architecture

When viewed as a library, not just an application, Synapse is made of up a few core components and concepts.

7.2.1 Library Architecture

The Synapse library is broken out in a hierarchical fashion. The root of the library contains application level code, such as the implementations of the Cortex, Axon, Cryotank, as well as the Telepath client and server components. There are also a set of common helper functions (common.py) and exceptions (exc.py). There are several submodules available as well:

synapse.cmds
Command implementations for the Cmdr CLI tool

synapse.data
Data files stored in the library.

synapse.lib
The lib module contains many of the primitives used by applications in order to implement them.

synapse.lookup
The lookup module contains various lookup definitions.

synapse.models
The models directory contains the core Synapse data model definitions.

synapse.servers
The servers module contains servers use to start and run Synapse applications.

synapse.tests
This is test code. It also contains a useful helper `synapse.tests.utils` which defines our base test class.

synapse.tools
The tools module contains various tools used to interact with the Synapse ecosystem.

synapse.vendor
This contains third-party code and associated LICENSE files. This is for internal library use only; no external API stability is guaranteed for any libraries under this module.

7.2.2 Object hierarchies

There is one base class that many objects inherit from, the `Base` (base.py) class. The `Base` class provides a few useful components (including, but not limited too):

- A way to do asynchronous object construction by override the `__anit__` method. This method is executed inside the python ioloop, allowing the object construction to do async function calls. An implementer still needs to call `await s_base.Base.__anit__(self)` first in order to ensure that the `Base` is setup properly.
- A way to register object teardown methods and perform object teardowns via the `onfini()` and `fini()`. These allow us to keep more granular control over how things are shut down and resources are released, versus relying solely on the garbage collector to handle teardowns properly. Often times, order matters, so we need to be sure that things are torn down cleanly. These routines can be registered during `__anit__`.
- `Base` objects are made via await the call to the `Base.anit()` function. If the `__anit__` function completed then the `anitted` attribute on the object will be True, otherwise it will be False.

- Context manager support. The `Base` object has native async context manager support, and upon exiting the context it will call `fini()` to do teardown. This pattern is convenient since it allows us to freely create `Base` classes without having to remember to always have to tear them down.
- The `Base` contains helpers for implementing an observable design pattern, where functions can be registered as event handlers, and events can be fired on the object at will. This can be very powerful for signaling across disparate components which would be otherwise too heavy to have explicit callbacks for.
- The `Base` contains helpers for executing asyncio coroutines on the ioloop. This is most commonly done via the `schedCoroTask` routine. This will schedule the coroutine to run on the ioloop, register the task with the `Base` and return the asyncio future. During `Base` fini, any coroutines still executing will be cancelled. This makes it very easy to schedule free-running coroutines from any `Base` class.

There are a few very important classes which use the `Base` object:

- The Synapse `Cell`. This is a batteries included primitive for running an application.
- The Telepath `Daemon`. This serves as a RPC server component.
- The Telepath `Proxy`. This serves as a RPC client component.

The `Cell` (cell.py) is a `Base` implementation which has several components available to it:

- It is a `Base`, so it benefits from all the components a `Base` has.
- It contains support for configuration directives at start time, so a cell can have well defined configuration options availble to it.
- It has persistent storage available via two different mechanisms, a LMDB slab for arbitrary data that is local to the cell, and a `Hive` for key-value data storage that can be remotely read and written.
- It handles user authentication and authorization via user data stored in the `Hive`.
- The `Cell` is Telepath aware, and will start his own `Daemon` that allows remote access. By default, the `Cell` has a PF Unix socket available for access, so local telepath access is trivial.
- Since the `Cell` is Telepath aware, there is a base `CellApi` that implements his RPC routines. `Cell` implementers can easily sublcass the `CellApi` class to add additional RPC routines.
- The `Cell` also contains hooks for easily starting a Tornado webserver. This allows us to trivially add web API routes to an object.
- The `Cell` contains a `Boss` which can be used to remotely enumerate and cancel managed coroutines.

Since the cell contains so much core management functionality, adding functionality to the Synapse `Cell` allows **all** applications using a Cell to be immediately extended to take advantage of that functionality without having to revisit multiple different implementations to update them. For this reason, our core application components (the `Axon`, `Cortex`, and `CryoCell`) all implement the `Cell` class. For example, if we add a new user management capability, that is now available to all those applications, as well as any others `Cell` implementations.

The application level components themselves have servers in the `synapse.servers` module, but there is also a generic server for starting any cell, `synapse.servers.cell`. These servers will create the `Cell`, and also add any additional RPC or HTTP API listening servers as necessary. Those are the preferred ways to run an application implemented via a `Cell`.

7.2.3 Telepath RPC

The Telepath RPC protocol is a lightweight RPC protocol used in Synapse. The server component, the previously mentioned `Daemon`, is used to share objects. An object may or may not be Telepath aware. In the case that it is not aware, all of its methods are exposed via Telepath. Objects which are Telepath aware, such as the `Cell`, implement an API interface that allows much more fine grained control over the the methods which are remotely available.

The base Telepath client is the `Proxy` class, this is used to connect to the Daemon. The `Proxy` intercepts attribute lookups to make and set remote method helpers at runtime, and sends those requests to the Daemon to be serviced. A *very* brief example of this is the following:

```
import synapse.telepath as s_telepath

url = 'tcp://user:secret@1.2.3.4:27492/someObject'

async with await s_telepath.openurl(url) as proxy:

    # Make attribute called "someMethod" on the proxy
    # then send a task to the server called "someMethod"
    # with the argument of somearg=1234
    resp = proxy.someMethod(somearg=1234)
    # The resp is the result of calling the someMethod argument on
    # the object named someObject on the daemon.
    print(resp)
```

A few notes about Telepath:

- Telepath remote call arguments and server responses must be able to be serialized using the msgpack protocol.
- Telepath supports generator protocols; so a server API may be a synchronous or asynchronous generator. From the proxy perspective, these are both considered asynchronous generators.
- The Telepath `Proxy` contains some helpers that allow is to be used from non-async code. These helpers run their API calls through the currently running ioloop, and will cause the client to make an ioloop if one is not currently running.
- Remote calls that raise exceptions on the server will have that exception serialized and sent back to the `Proxy`. The `Proxy` will then raise an exception to the caller.
- Methods calls prefixed with a underscore (`_somePrivatMethod()` for example) will be rejected by the `Daemon`. This does allow us to protect private methods on shared objects.

7.3 Cortex Development Quickstart

This guide is intended for developers looking to integrate Synapse components with other applications by using the Telepath API. Additionally, this guide will introduce developers to writing custom Cortex modules in Python to allow custom data model extensions, storm commands, ingest functions, and change hooks. This guide assumes familiarity with deploying Cortex servers and the Storm query syntax. For help on getting started, see *Getting Started*.

For complete API documentation on all Synapse components see *Synapse Python API*.

7.3.1 Remote Cortex Access

A Cortex, like most synapse components, provides two mechanisms for remote API calls. The HTTP/REST API and the Telepath API. For additional documentation on the Cortex HTTP API, see *Synapse HTTP/REST API*. This guide will cover remote API calls using Telepath.

Telepath is an asynchronous, high-performance, streaming oriented, RPC protocol. It is designed for minimum development effort and maximum performance. Data is serialized using the highly efficient Message_Pack format which is not only more size efficient than JSON, but allows serialization of binary data and supports incremental decoding for use in stream based protocols.

Telepath allows a client to connect to a Python object shared on a remote server and, in most instances, call methods as though the object were local. However, this means all arguments and return values must be serializable using Message Pack.

To connect to a remote object, the caller specifies a URI to connect and construct a Telepath Proxy. In the following examples, we will assume a Cortex was previously setup and configured with the user `visi` and the password `secretsauce` running on port `27492` on the host `1.2.3.4`.

Making a simple call

Once a Telepath proxy is connected, most methods may simply be called as though the object were local. For example, the `getModelDict` method on the `CoreApi` returns a Python dictionary containing the details of the data model in the remote Cortex.

```
import asyncio
import synapse.telepath as s_telepath

async def main():

    async with await s_telepath.openurl('tcp://visi:secretsauce@1.2.3.4:27492/') as core:

        model = await core.getModelDict()

        for form in model.get('forms'):
            dostuff()

if __name__ == '__main__':
    asyncio.run(main())
```

Like many objects in the Synapse ecosystem, a Telepath proxy inherits from `synapse.lib.base.Base`. This requires the `fini` method to be called to release resources and close sockets. In the example above, we use the async context manager implemented by the `Base` class (`async with`) to ensure that the proxy is correctly shutdown. However, Telepath is designed for long-lived Proxy objects to minimize API call delay by using existing sockets and sessions. A typical app will create a telepath proxy during initialization and only create a new one in the event that the remote Telepath server is restarted.

The above example also demonstrates that Telepath is designed for use with Python 3.11 asyncio. However, the Telepath proxy can also be constructed and used transparently from non-async code as seen below.

```
import synapse.telepath as s_telepath

def main():

    core = s_telepath.openurl('tcp://visi:secretsauce@1.2.3.4:27492/')
```

(continues on next page)

(continued from previous page)

```
    model = core.getModelDict()

if __name__ == '__main__':
    main()
```

The remainder of the examples in this guide will assume the use of an asyncio loop.

Generators and Yielding

Many of the Telepath APIs published by Synapse services are capable of yielding results as a generator to facilitate incremental reads and time_to_first_byte (TTFB) optimizations. In the remote case, this means the caller may receive and begin processing results before all of the results have been enumerated by the server. Any Python async generator method on a shared object may be iterated by a client with full back_pressure to the server. This means a caller may issue a query which produces a very large result set and consume the results incrementally without concern over client/server memory exhaustion due to buffering. The following example demonstrates using the Cortex `storm` API to retrieve a message stream, which includes nodes in it.

```
import asyncio
import synapse.telepath as s_telepath

async def main():

    async with await s_telepath.openurl('tcp://visi:secretsauce@1.2.3.4:27492/') as core:

        async for mesg in core.storm('inet:ipv4 | limit 10000'):

            # Handle node messages specifically.
            if mesg[0] == 'node':
                node = mesg[1]
                dostuff(node)

            else:
                # Handle non-node messages.
                do_not_node_stuff(mesg)

if __name__ == '__main__':
    asyncio.run(main())
```

The `storm()` API is the preferred API to use for executing Storm queries on a Cortex. It generates a series of messages which the caller needs to consume.

For API documentation on the full Cortex Telepath API, see CoreAPi.

7.4 Synapse Docker Builds

This doc details the docker builds and scripts used by Synapse.

7.4.1 Images

There are several images provided by the Synapse repository. These are built from an external image that is periodically updated with core Synapse dependencies.

The images provided include the following:

vertexproject/synapse
: This container just contains Synapse installed into it. It does not start any services.

vertexproject/synapse-aha
: This container starts the Aha service.

vertexproject/synapse-axon
: This container starts the Axon service.

vertexproject/synapse-cortex
: This container starts the Cortex service.

vertexproject/synapse-cryotank
: This container starts the Cryotank service.

vertexproject/synapse-jsonstor
: This container starts the JSONStor service.

vertexproject/synapse-stemcell
: This container launches the Synapse stemcell server.

7.4.2 Building All Images

Images are built using Bash scripts. All of the images can be built directly with a single command:

```
$ ./docker/build_all.sh <optional_image_tag>
```

If the image tag is not provided, it will tag the images with `:dev_build`.

7.4.3 Building a Specific Application Image

A specific application images can be built as well.

```
$ ./docker/build_image.sh <application> <optional_image_tag>

# Example of building a local Cortex image.

$ ./docker/build_image.sh cortex my_test_image
```

If the image tag is not provided, it will tag the image with `:dev_build`.

7.4.4 Building the `vertexproject/synapse` image

The bare image with only Synapse installed on it can be built like the following:

```
$ docker build --progress plain --pull -t vertexproject/synapse:$TAG -f docker/
↪images/synapse/Dockerfile .

# Example of building directly with the tag mytag

$ docker build --progress plain --pull -t vertexproject/synapse:mytag -f␣
↪docker/images/synapse/Dockerfile .
```

7.4.5 Working with Synapse Images

Developers working with Synapse images should consider the following items:

- The Synapse images are not locked to a specific Python version. The underlying Python minor version or base distribution may change. If they do change, that will be noted in the Synapse changelog. If you are building containers off of a floating tag such as `vertexproject/synapse:v2.x.x`, make sure you are reviewing our *Synapse Changelog* for items which may affect your use cases. Python patch level updates will not be included in the changelogs.
- The `synapse` package, and supporting packages, are currently installed to the distribution Python environment. The version of `pip` installed in the containers is PEP668 aware. If you are installing your own Python packages to the distribution Python environment with ``pip`, you will need to add the `--break-system-packages` argument:

```
python -m pip install --break-system-packages yourTargetPackage
```

7.4.6 Verifying container image signatures

Synapse docker images which are release tagged (e.g. `v2.1.3` or `v2.x.x`) are accompanied with cosign signatures which can be used to assert that the image was produced by The Vertex Project. Branch builds, such as development `master` tags are not guaranteed to be signed.

You can use the Python script `synapse.tools.docker.validate` to confirm that a given image has a `cosign` signature which was signed by a Vertex Project code signing certificate; and then confirm that the `cosign` signature was signed by the certificate. This does require having `cosign` version v2.x.x available.

The following shows an example of verifying a signed image, referenced by its registry content hash:

```
$ python -m synapse.tools.docker.validate vertexproject/
↪synapse@sha256:4ec5d97e1bbdb49971f5c1d520a81371021ef4c84f932d9ef23a635a099cb53b
Verifying: vertexproject/
↪synapse@sha256:4ec5d97e1bbdb49971f5c1d520a81371021ef4c84f932d9ef23a635a099cb53b
Using Cosign with GitVersion:    v2.1.0
Loading certdir from /your/python/site-lib/synapse/synapse/data/certs
Verified certificate embedded in the signature.
Cosign output:
[{'critical': {'identity': {'docker-reference': 'index.docker.io/vertexproject/synapse'},
               'image': {'docker-manifest-digest':
↪'sha256:4ec5d97e1bbdb49971f5c1d520a81371021ef4c84f932d9ef23a635a099cb53b'},
               'type': 'cosign container image signature'},
```

(continues on next page)

(continued from previous page)

```
  'optional': {'Subject': '',
               'commit': '07ab34bbeb04cf0e96e7ba7b65c5a7d7baeda151',
               'jobid': '9b78fab5-d364-43b3-8fb0-3f0c82a795cd'}}]
Verified: vertexproject/
↪synapse@sha256:4ec5d97e1bbdb49971f5c1d520a81371021ef4c84f932d9ef23a635a099cb53b
```

Devops teams can use this tool as the basis to create or update an admissions controller that can ensure that only signed images are allowed to be used in their environments.

7.5 Storm Service Development

7.5.1 Anatomy of a Storm Service

A Storm Service (see *Service*) is a standalone application that extends the capabilities of the Cortex. One common use case for creating a service is to add a Storm command that will query third-party data, translate the results into the Synapse datamodel, and then ingest them into the hypergraph.

In order to leverage core functionalities it is recommended that Storm services are created as Cell implementations, and the documentation that follows will assume this. For additional information see *Synapse Architecture*.

A Storm service generally implements the following components:

- A *Package* that contains the new *Storm Service Commands* and optional new *Storm Service Modules*.
- A subclass of `synapse.lib.CellApi` which uses the `synapse.lib.StormSvc` mixin and contains the following information:
  - The service name, version, packages, and events as defined in `synapse.lib.StormSvc`.
  - Custom methods which will be accessible as Telepath API endpoints, and therefore available for use within defined Storm commands.
- A subclass of `synapse.lib.Cell` which includes additional configuration definitions and methods required to implement the service.

When implemented as a Cell, methods can also optionally have custom permissions applied to them. If a specific rule is added it should be namespaced with the service name, e.g. `svcname.rule1`. Alternatively, a method can wrapped with `@s_cell.adminapi()` to only allow admin access.

For additional details see *Minimal Storm Service Example*.

Connecting a service

Before connecting a service to a Cortex it is a best practice to add a new service user, which can be accomplished with `synapse.tools.cellauth`. For example:

```
python -m synapse.tools.cellauth tcp://root:<root_passwd>@<svc_ip>:<svc_port> modify␣
↪svcuser1 --adduser
python -m synapse.tools.cellauth tcp://root:<root_passwd>@<svc_ip>:<svc_port> modify␣
↪svcuser1 --passwd secret
```

If the service requires specific permissions for a new user they can also be added:

```
python -m synapse.tools.cellauth tcp://root:<root_passwd>@<svc_ip>:<svc_port> modify␣
↪svcuser1 --addrule svcname.rule1
```

Permissions to access the service can be granted by adding the `service.get.<svc_iden>` rule to the appropriate users / roles in the Cortex.

A Storm command can be run on the Cortex to add the new service, and the new service will now be present in the service list and Storm `help`.

Services are added to a Cortex with the `service.add` command.

```
storm> service.add mysvc tcp://root:secret@127.0.0.1:34867/
```

```
added bedc2951767542e1f668cf4be12b6eb3 (mysvc): tcp://root:secret@127.0.0.1:34867/
```

```
complete. 0 nodes in 42 ms (0/sec).
```

Services that have been connected to the Cortex can be listed with the `service.list` command.

```
storm> service.list
```

```
Storm service list (iden, ready, name, service name, service version, url):
```

```
bedc2951767542e1f668cf4be12b6eb3 false (mysvc) (Unknown @ Unknown): tcp://
↪root:secret@127.0.0.1:34867/
```

```
1 services
```

```
complete. 0 nodes in 167 ms (0/sec).
```

7.5.2 Storm Service Commands

Implementation

Multiple Storm commands can be added to a Storm service package, with each defining the following attributes:

- `name`: Name of the Storm command to expose in the Cortex.
- `descr`: Description of the command which will be available in `help` displays.
- `cmdargs`: An optional list of arguments for the command.
- `cmdconf`: An optional dictionary of additional configuration variables to provide to the command Storm execution.
- `forms`: List of input and output forms for the command.
- `storm`: The Storm code, as a string, that will be executed when the command is called.

Typically, the Storm code will start by getting a reference to the service via `$svc = $lib.service.get($cmdconf.sviden)` and reading in any defined `cmdargs` that are available in `$cmdopts`. The methods defined in the service's Cell API can then be called by, for example, `$retn = $svc.mysvcmethod($cmdopts.query)`.

Input/Output Conventions

Most commands that enrich or add additional context to nodes should simply yield the nodes they were given as inputs. If they don't know how to enrich or add additional context to a given form, nodes of that form should be yielded rather than producing an error. This allows a series of enrichment commands to be pipelined regardless of the different inputs that a given command knows how to operate on.

Argument Conventions

`--verbose`

In general, Storm commands should operate silently over their input nodes and should especially avoid printing anything "per node". However, when an error occurs, the command may use `$lib.warn()` to print a warning message per-node. Commands should implement a `--verbose` command line option to enable printing "per node" informational output.

`--debug`

For commands where additional messaging would assist in debugging a `--debug` command line option should be implemented. For example, a Storm command that is querying a third-party data source could use `$lib.print()` to print the raw query string and raw response when the `--debug` option is specified.

`--yield`

For commands that create additional nodes, it may be beneficial to add a `--yield` option to allow a query to operate on the newly created nodes. Some guidelines for `--yield` options:

- The command should *not* yield the input node(s) when a `--yield` is specified
- The `--yield` option should *not* be implemented when pivoting from the input node to reach the newly created node is a "refs out" or 1-to-1 direct pivot. For example, there is no need to have a `--yield` option on the `maxmind` command even though it may create an `inet:asn` node for an input `inet:ipv4` node due to the 1-to-1 pivot -> `inet:asn` being possible.
- The `--yield` option should ideally determine a "primary" node form to yield even when the command may create many forms in order to tag them or update .seen times.

7.5.3 Storm Service Modules

Modules can be added to a Storm service package to expose reusable Storm functions. Each module defines a `name`, which is used for importing elsewhere via `$lib.import()`, and a `storm` string. The Storm code in this case contains callable functions with the format:

```
function myfunc(var1, var2) {
    // function Storm code
}
```

7.5.4 Minimal Storm Service Example

A best practice is to separate the Storm and service code into separate files, and nest within a `synmods` directory to avoid Python namespace conflicts:

```
service-example
├── synmods
│   └── example
│       ├── __init__.py
│       ├── service.py
│       ├── storm.py
│       └── version.py
```

The Storm package and the service should also maintain consistent versioning.

For convenience, the example below shows the Storm code included in the `service.py` file.

service.py

```
import sys
import asyncio

import synapse.lib.cell as s_cell
import synapse.lib.stormsvc as s_stormsvc

# The Storm definitions below are included here for convenience
# but are typically contained in a separate storm.py file and imported to service.py.
# Other Storm commands could be created to call the additional Telepath endpoints.
svc_name = 'example'
svc_guid = '0ecc1eb65659a0f07141bc1a360abda3'  # can be generated with synapse.common.
↪guid()
svc_vers = (0, 0, 1)

svc_evts = {
    'add': {
        'storm': f'[(meta:source={svc_guid} :name="Example data")]'
    }
}

svc_mod_ingest_storm = '''
function ingest_ips(data, srcguid) {
    $results = $lib.set()

    for $ip in $data {
        [ inet:ipv4=$ip ]

        // Lightweight edge back to meta:source
        { [ <(seen)+ { meta:source=$srcguid } ] }

        { +inet:ipv4 $results.add($node) }
    }

    | spin |
```

(continues on next page)

(continued from previous page)

```
    return($results)
}
'''

# The first line of this description will display in the Storm help
svc_cmd_get_desc = '''
Query the Example service.

Examples:

    # Query the service and create an IPv4 node
    inet:fqdn=good.com | example.get

    # Query the service and yield the created inet:ipv4 node
    inet:fqdn=good.com | example.get --yield
'''

svc_cmd_get_forms = {
    'input': [
        'inet:fqdn',
    ],
    'output': [
        'inet:ipv4',
    ],
}

svc_cmd_get_args = (
    ('--yield', {'default': False, 'action': 'store_true',
                 'help': 'Whether to yield the created nodes to the output stream.'}),
    ('--debug', {'default': False, 'action': 'store_true',
                 'help': 'Enable debug output.'}),
)

svc_cmd_get_conf = {
    'srcguid': svc_guid,
}

svc_cmd_get_storm = '''
init {
    $svc = $lib.service.get($cmdconf.svciden)
    $ingest = $lib.import(example.ingest)
    $srcguid = $cmdconf.srcguid
    $debug = $cmdopts.debug
    $yield = $cmdopts.yield
}

// $node is a special variable that references the inbound Node object
$form = $node.form()

switch $form {
    "inet:fqdn": {
```

(continues on next page)

(continued from previous page)

```
        $query=$node.repr()
    }
    *: {
        $query=""
        $lib.warn("Example service does not support {form} nodes", form=$form)
    }
}

// Yield behavior to drop the inbound node
if $yield { spin }

// Call the service endpoint and ingest the results
if $query {
    if $debug { $lib.print("example.get query: {query}", query=$query) }

    $retn = $svc.getData($query)

    if $retn.status {
        $results = $ingest.ingest_ips($retn.data, $srcguid)

        if $yield {
            for $result in $results { $lib.print($result) yield $result }
        }
    } else {
        $lib.warn("example.get error: {err}", err=$retn.mesg)
    }
}
'''

svc_cmds = (
    {
        'name': f'{svc_name}.get',
        'descr': svc_cmd_get_desc,
        'cmdargs': svc_cmd_get_args,
        'cmdconf': svc_cmd_get_conf,
        'forms': svc_cmd_get_forms,
        'storm': svc_cmd_get_storm,
    },
)

svc_pkgs = (
    {
        'name': svc_name,
        'version': svc_vers,
        'synapse_version': '>=2.144.0,<3.0.0',
        'modules': (
            {
                'name': f'{svc_name}.ingest',
                'storm': svc_mod_ingest_storm,
            },
        ),
        'commands': svc_cmds,
```

(continues on next page)

(continued from previous page)

```
    },
)

class ExampleApi(s_cell.CellApi, s_stormsvc.StormSvc):
    '''
    A Telepath API for the Example service.
    '''

    # These defaults must be overridden from the StormSvc mixin
    _storm_svc_name = svc_name
    _storm_svc_vers = svc_vers
    _storm_svc_evts = svc_evts
    _storm_svc_pkgs = svc_pkgs

    async def getData(self, query):
        return await self.cell.getData(query)

    async def getInfo(self):
        await self._reqUserAllowed(('example', 'info'))
        return await self.cell.getInfo()

    @s_cell.adminapi()
    async def getAdminInfo(self):
        return await self.cell.getAdminInfo()

class Example(s_cell.Cell):

    cellapi = ExampleApi

    confdefs = {
        'api_key': {
            'type': 'string',
            'description': 'API key for accessing an external service.',
        },
        'api_url': {
            'type': 'string',
            'description': 'The URL for an external service.',
            'default': 'https://example.com',
        },
    }

    async def __anit__(self, dirn, conf):
        await s_cell.Cell.__anit__(self, dirn, conf=conf)
        self.apikey = self.conf.get('api_key')
        self.apiurl = self.conf.get('api_url')

    async def getData(self, query):
        # Best practice is to also return a status and optional message in case of an␣
↪error
        retn = {
            'status': True,
            'data': None,
```

(continues on next page)

(continued from previous page)

```
            'mesg': None,
        }

        # Retrieving and parsing data would go here
        if query == 'good.com':
            data = ['1.2.3.4', '5.6.7.8']
            retn['data'] = data

        else:
            retn['status'] = False
            retn['mesg'] = 'An error occurred during data retrieval.'

        return retn

    async def getInfo(self):
        info = {
            'generic': 'info',
        }

        return info

    async def getAdminInfo(self):
        info = {
            'admin': 'info',
        }

        return info
```

7.6 Storm API Guide

7.6.1 Storm APIs

Storm is available over Telepath and HTTP API interfaces. Both interfaces require a Storm query string, and may take additional `opts` arguments.

Telepath

There are three Storm APIs exposed via Telepath.

storm(text, opts=None)
The Storm API returns a message stream. It can be found here storm.

callStorm(text, opts=None)
The callStorm API returns a message given by the Storm `return( )` syntax. It can be found here callStorm.

count(text, opts=None)
The count API returns a count of the number of nodes which would have been emitted by running a given query. It can be found here *Cortex*.

HTTP API

The HTTP API versions of the Storm APIs can be found here Cortex HTTP API.

`/v1/api/storm`
This API returns a message stream.

`/v1/api/storm/call`
This API returns a message given by the Storm `return( )` syntax.

`/v1/api/storm/export`
This API returns a stream of msgpack encoded data, which can be used as a *.nodes* file for later import.

7.6.2 Message Types

The Telepath `storm()` and HTTP `api/v1/storm` APIs yield messages from the Storm runtime to the caller. These are the messages that may be seen when consuming the message stream.

Each message has the following basic structure:

```
[ "type", { ..type specific info... } ]
```

init

First message sent by a Storm query runtime.

It includes the following keys:

task
The task identifier (which can be used for task cancellation).

tick
The epoch time the query execution started (in milliseconds). This value is computed from the host time and may be affected by any changes in the host clock.

abstick
The relative time that the query execution started (in milliseconds). This value is computed from a monotonic clock and can be used as a reference time.

text
The Storm query text.

hash
The md5sum of the Storm query text.

Example:

```
('init',
 {'task': '8c90c67e37a30101a2f6a7dfb2fa0805',
  'text': '.created | limit 3',
  'hash': '2d16e12e80be53e0e79e7c7af9bda12b',
  'tick': 1539221678859})
```

node

This represents a packed node. Each serialized node will have the following structure:

```
[
    [<form>, <valu>],       # The [ typename, typevalue ] definition of the node.
    {
        "iden": <hash>,     # A stable identifier for the node.
        "tags": {},         # The tags on the node.
        "props": {},        # The node's secondary properties.
        "path": {},         # Path related information in the node.
        "tagprops": {},     # The node's tag properties.

        # optional
        "repr": ...         # Presentation values for the type value.
        "reprs": {}         # Presentation values for props which need it.
        "tagpropreprs": {}  # Presentation values for tagprops which need it.

    }
]
```

Example:

This example is very simple - it does not include repr information, or things related to path data:

```
('node',
 (('inet:fqdn', 'icon.torrentart.com'),
  {'iden': 'ae6d871163980f82dc1d3b06e784a80e8085493f68fbf2813c9681cb3e2630a8',
   'props': {'.created': 1526590932444,
             '.seen': (1491771661000, 1538477660797),
             'domain': 'torrentart.com',
             'host': 'icon',
             'issuffix': 0,
             'iszone': 0,
             'zone': 'torrentart.com'},
   'tags': {'aka': (None, None),
            'aka.beep': (None, None),}}))
```

For path and repr information, see the examples in the opts documentation *Storm Opts*.

ping

A keepalive message. This is sent periodically when the `keepalive` options is set. See *Storm Opts* for more information.

print

The print event contains a message intended to be displayed to the caller.

It includes the following key:

mesg
The message to be displayed to the user.

Example:

```
(print, {'mesg': 'I am a message!'})
```

This can be produced by users with the `$lib.print()` Storm API.

warn

The warn event contains data about issues encountered when performing an action.

It includes the following keys:

mesg
The message to be displayed to the user.

The warn event may contain additional, arbitrary keys in it.

Example:

```
('warn',
 {'mesg': 'Unable to foo the bar.com domain',
  'domain': 'bar.com'})
```

This can be produced by users with the `$lib.warn()` Storm API.

err

The err event is sent if there is a fatal error encountered when executing a Storm query. There will be no further processing; only a `fini` message sent afterwards.

The err event does contain a marshalled exception in it. This contains the exception type as the identifier; and several attributes from the exception.

The following keys are usually present in the marshalled information:

esrc
Source line that raised the exception.

efile
File that the exception was raised from.

eline
Line number from the raising file.

ename
Name of the function where the exception was from.

mesg
The `mesg` argument to a SynErr exception, if present; or the `str()` exception.

Additional keys may also be present, depending on the exception that was raised.

Example:

```
('err',
 ('BadTypeValu',
  {'efile': 'inet.py',
   'eline': 294,
   'form': 'inet:fqdn',
   'mesg': 'FQDN failed to match fqdnre [^[\\w._-]+$]',
   'name': 'inet:fqdn',
   'valu': '1234@#'}))
```

fini

The last message sent by a Storm query runtime. This can be used as a key to stop processing messages or finalize any sort of rollup of messages.

It includes the following keys:

tock
The epoch time the query execution finished (in milliseconds). This value is computed from adding the `took` value to the `tick` value from the `init` message.

took
The amount of time it took for the query to execute (in milliseconds). This value is computed from the `abstick` and `abstock` values.

abstock
The relative time that the query execution finished at (in milliseconds). This value is computed from a monotonic clock and should always be equal to or greater than the `abstick` value from the `init` message.

count
The number of nodes yielded by the runtime.

Example:

```
('fini', {'count': 1, 'tock': 1539221715240, 'took': 36381})
```

Note: If the Storm runtime is cancelled for some reason, there will be no `err` or `fini` messages sent. This is because the task cancellation may tear down the channel and we would have an async task blocking on attempting to send data to a closed channel.

node:edits

The `node:edits` message represents changes that are occurring to the underlying graph, as a result of running a Storm query.

It includes the following key:

edits
A list of changes made to a set of nodes.

Example:

```
# Nodeedits produced by the following query: [(inet:ipv4=1.2.3.4 :asn=1)]

('node:edits',
 {'edits': (('20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f',
             'inet:ipv4',
             ((0, (16909060, 4), ()),
              (2, ('.created', 1662578208195, None, 21), ()),
              (2, ('type', 'unicast', None, 1), ()))),)})
('node:edits',
 {'edits': (('20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f',
             'inet:ipv4',
             ((2, ('asn', 1, None, 9), ()),)),
            ('371bfbcd479fec0582d55e8cf1011c91c97f306cf66ceea994ac9c37e475a537',
             'inet:asn',
             ((0, (1, 9), ()),
              (2, ('.created', 1662578208196, None, 21), ()))))})
```

node:edits:count

The `node:edits:count` message represents a summary of changes that are occurring to the underlying graph, as a result of running a Storm query. These are produced when the query `opts` set `editformat` to `count`.

It includes the following key:

count
The number of changes made to the graph as a result of a single `node:edits` event.

Example:

```
# counts produced by the following query: [(inet:ipv4=1.2.3.4 :asn=1)]

('node:edits:count', {'count': 3})
('node:edits:count', {'count': 3})
```

storm:fire

The `storm:fire` message is a arbitrary user created message produced by the `$lib.fire()` Storm API. It includes the following keys:

type
The type of the event.

data
User provided data.

Example:

```
# The following query produces an event
$l = ((1), (2), (3)) $lib.fire('demo', key=valu, somelist=$l)

# The event produced.
('storm:fire', {'data': {'key': 'valu', 'somelist': (1, 2, 3)}, 'type': 'demo'})
```

look:miss

The `look:miss` message is sent when the Storm runtime is set to `lookup` mode and the node that was identified by the scrape logic is not present in the current View.

It includes the following key:

ndef
A tuple of the form and normalized value.

Example:

```
('look:miss', {'ndef': ('inet:fqdn', 'hehe.com')})

# The ipv4 value is presented in system mode.
('look:miss', {'ndef': ('inet:ipv4', 16909060)})
```

csv:row

The `csv:row` message is sent by the Storm runtime by the `$lib.csv.emit()` Storm API.

It includes the following keys:

row
A list of elements that make up the row.

table
A optional table name. This may be `None`.

Example:

```
# This query produces the following event: $lib.csv.emit(foo, bar, $lib.time.now())
('csv:row', {'row': ('foo', 'bar', 1662578057658), 'table': None})

# This query produces the following event: $lib.csv.emit(foo, bar, $lib.time.now(),␣
↪table=foo)
('csv:row', {'row': ('foo', 'bar', 1662578059282), 'table': 'foo'})
```

7.6.3 Storm Call APIs

The Telepath `callStorm()` and HTTP API `storm/call` interfaces are designed to return a single message to the caller, as opposed to a stream of messages. This is done using the Storm `return( )` syntax. Common uses for the call interfaces include getting and setting values where the full message stream would not be useful.

Example:

The following example shows retrieving a user definition.

```
# Prox is assumed to be a Telepath proxy to a Cortex.
>>> text = '$user = $lib.auth.users.byname($name) return ( $user )'
>>> opts = {'vars': {'name': 'root'}}
>>> ret = prox.callStorm(text, opts=opts)
>>> pprint(ret)
{'admin': True,
 'archived': False,
 'authgates': {'0b942d5f4309d70e5fa64423714e25aa': {'admin': True},
               'cdf6f1727da73dbac95e295e5d258847': {'admin': True}},
 'email': None,
 'iden': '933a320b7ce8134ba5abd93aa487e1b5',
 'locked': False,
 'name': 'root',
 'roles': (),
 'rules': (),
 'type': 'user'}
```

The following shows setting an API key for a Power-Up. There is no `return` statement, so the return value defaults to None.

```
# Prox is assumed to be a Telepath proxy to a Cortex.
>>> text = 'foobar.setup.apikey $apikey'
>>> opts = {'vars': {'apikey': 'secretKey'}}
>>> ret = prox.callStorm(text, opts=opts)
>>> print(ret)
None
```

7.6.4 Storm Opts

All Storm API endpoints take an `opts` argument. This is a dictionary that contains metadata that is used by the Storm runtime for various purposes. Examples are given using Python syntax.

debug

If this is set to True, the Storm runtime will be created with `$lib.debug` set to True.

Example:

```
opts = {'debug': True}
```

editformat

This is a string containing the format that node edits are streamed in. This may be `nodeedits` (the default value), `none`, or `count`. If the value is `none`, then no edit messages will be streamed. If the value is `count`, each `node:edits` message is replaced by a `node:edits:count` message, containing a summary of the number of edits made for a given message.

Examples:

```
# Turn node:edit messages into counts
opts = {'editformat': 'count'}

# Disable node edits
opts = {'editformat': 'none'}
```

idens

This is a list of node iden hashes to use as initial input to the Storm runtime. These nodes are lifted after any `ndefs` options are lifted, but prior to regular lift operations which may start a Storm query.

Example:

```
idens = ('ee6b92c9fd848a2cb00f3a3618148c512b58456b8b51fbed79251811597eeea3',
         'c5a67a095b71771d9663d691f0ab36b53ebdc14fbad18f23f95e923543156bd6',)
opts = {'idens': idens}
```

keepalive

This is the period (in seconds) in which to send a `ping` message from a Storm query which is streamiing results, such as the Telepath `.storm()` API or the HTTP `/v1/api/storm` API endpoint. This may be used with long-running Storm queries when behind a network proxy or load balancer which may terminate idle connections.

The keepalive value must be greater than zero.

Example:

```
keepalive = 2  # Send a keepalive message every 2 seconds
opts = {'keepalive': keepalive}
```

limit

Limit the total number of nodes that the Storm runtime produces. When this number is reached, the runtime will be stopped.

Example:

```
opts = {'limit': 100}
```

mode

This is the mode that a Storm query is parsed in. This value can be specified to `lookup`, `autoadd`, and `search` modes to get different behaviors.

Example:

```
# Using lookup mode, the query text, before switching to command mode with a |␣
↪character,
# will have its text scrapped for simple values such as FQDNs, IP Addresses,␣
↪and Hashes
# and attempt to lift any matching nodes.
opts = {'mode': 'lookup'}

# Using autoadds mode, the query text is scrapped like in lookup mode; and for␣
↪any
# values which we try to lift that do not produce nodes, those nodes will be␣
↪added
# in the current view.
opts = {'mode': 'autoadd'}

# Using search mode, the query will be run through the Storm search interface.
# This will lift nodes based on searching, which is enabled by the
# Synapse-Search Advanced Power-up.
opts = {'mode': 'search'}
```

ndefs

This is a list of form and value tuples to use as initial input to the Storm runtime. These are expected to be the already normalized, system mode, values for the nodes. These nodes are lifted before any other lift operators are run.

Example:

```
ndefs = (
    ('inet:fqdn', 'com'),
    ('inet:ipv4', 134744072),
)

opts = {'ndefs': ndefs}
```

nexsoffs

Wait for the Cortex to reach the specified Nexus offset before executing the query. This is intended for internal use when offloading queries to a mirror.

Example:

```
opts = {'nexsoffs': 7759195}
```

nexstimeout

Timeout (in seconds) to wait for the Cortex to reach the Nexus offset specified by `nexsoffs`. This is intended for internal use when offloading queries to a mirror.

Example:

```
opts = {'nexstimeout': 5}
```

mirror

If Storm query offloading is configured and `mirror` is `true` (the default value), the Cortex will attempt to offload the query to a mirror. Setting this to `false` will force the query to be run on the local Cortex rather than a mirror.

Example:

```
opts = {'mirror': False}
```

path

If this is set to True, the `path` key in the packed nodes will contain a `nodes` key, which contains a list of the node iden hashes that were used in pivot operations to get to the node.

Example:

```
opts = {'path': True}

# A Storm node message with a node path added to it, from the query inet:ipv4 ->␣
↪inet:asn.

('node',
 (('inet:asn', 1),
  {'iden': '371bfbcd479fec0582d55e8cf1011c91c97f306cf66ceea994ac9c37e475a537',
   'nodedata': {},
   'path': {'nodes': ('20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f',
                      '371bfbcd479fec0582d55e8cf1011c91c97f306cf66ceea994ac9c37e475a537
↪')},
   'props': {'.created': 1662493825668},
   'tagprops': {},
   'tags': {}}))
```

readonly

Run the Storm query in a readonly mode. This prevents editing the graph data, and only allows a small subset of whitelisted Storm library functions to be used.

Examples:

```
opts = {'readonly': True}
```

repr

If this is set to True, the packed node will have a `repr` and `reprs` key populated, to contain human friendly representations of system mode values.

Example:

```
opts = {'repr': True}

# A Storm node message with reprs added to it.

('node',
 (('inet:ipv4', 134744072),
  {'iden': 'ee6b92c9fd848a2cb00f3a3618148c512b58456b8b51fbed79251811597eeea3',
   'nodedata': {},
   'path': {},
   'props': {'.created': 1662491423034, 'type': 'unicast'},
   'repr': '8.8.8.8',
   'reprs': {'.created': '2022/09/06 19:10:23.034'},
   'tagpropreprs': {},
   'tagprops': {},
   'tags': {}}))
```

scrub

This is a set of rules that can be provided to the Storm runtime which dictate which data should be included or excluded from nodes that are returned in the message stream. Currently the only rule type supported is `include` for `tags`.

Example:

```
# Only include tags which start with cno and rep.foo
scrub = {'include': {'tags': ['cno', 'rep.foo',]}}
opts = {'scrub': scrub}

# Do not include any tags in the output
scrub = {'include': {'tags': []}}
opts = {'scrub': scrub}
```

show

A list of message types to include in the output message stream. The `init`, `fini`, and `err` message types cannot be filtered with this option.

Example:

```
# Only node and warning messages.
opts = {'show': ['node', 'warning']}

# Only include required messages.
opts = {'show': []}
```

show:storage

A boolean option which, when set to `true`, instructs the Storm runtime to add a `storage` key to each yielded node which contains a raw breakdown of storage nodes which can be used to determine which parts of the node are stored in which layer within the view.

Example:

```
opts = {'show:storage': True}
```

task

A user provided guid that is used as the task identifier for the Storm runtime. This allows a user to have a predictable identifier that they can use for task cancellation.

The Storm runtime will raise a `BadArg` value if the `task` iden is associated with a currently running task.

Example:

```
# Generate a guid on the client side and provide it to the Cortex
import synapse.common as s_commmon
task_iden = s_common.guid()
opts = {'task': task_iden}
```

user

The User iden to run the Storm query as. This allows a user with the permission `impersonate` to run a Storm query as another user.

Example:

```
opts = {'user': 6e9c8de2f1aa39fee11c19d0974e0917}
```

vars

A dictionary of key - value pairs that are mapped into the Storm runtime as variables. Some uses of this include providing data to the runtime that is used with an ingest script, or to provide secrets to the Storm runtime so that they will not be logged.

Example:

```
# A secret key - A good example of this is configuring a Rapid Power-Up.
vars = {'secretkey': 'c8de2fe11c19d0974e091aa39fe176e9'}
opts = {'vars': vars}

# Some example data that could be used in a Storm ingest script.
records = (
    ('foobar.com', '8.8.8.8', '20210810'),
    ('bazplace.net', '1.2.3.4', '20210810'),
)
vars = {'records': records}
opts = {'vars': vars}
```

view

The View iden in which to run the Storm query in. If not specified, the query will run in the user's default view.

Example:

```
opts = {'view': 31ded629eea3c7221be0a61695862952}
```

CHAPTER

EIGHT

SYNAPSE GLOSSARY

This Glossary provides a quick reference for common terms related to Synapse technical and analytical concepts.

8.1 A

8.1.1 Addition, Automatic

See *Autoadd*.

8.1.2 Addition, Dependent

See *Depadd*.

8.1.3 Advanced Power-Up

See *Power-Up, Advanced*.

8.1.4 Admin Tool

See *Tool, Admin*.

8.1.5 Analytical Model

See *Model, Analytical*.

8.1.6 Auth Gate

An auth gate (short for "authorization gate", informally a "gate") is an object within a *Service* that may have its own set of permissions.

Both a *Layer* and a *View* are common examples of auth gates.

8.1.7 Autoadd

Short for "automatic addition". Within Synapse, a feature of node creation where any secondary properties that are derived from a node's primary property are automatically set when the node is created. Because these secondary properties are based on the node's primary property (which cannot be changed once set), the secondary properties are read-only.

For example, creating the node `inet:email=alice@mail.somecompany.org` will result in the autoadd of the secondary properties `inet:email:user=alice` and `inet:email:domain=mail.somecompany.org`.

See also the related concept *Depadd*.

8.1.8 Axon

The Axon is a *Synapse Service* that provides binary / blob ("file") storage within the Synapse ecosystem. An Axon indexes binaries based on their SHA-256 hash for deduplication. The default Axon implemenation stores the blobs in an LMDB *Slab*.

8.2 B

8.2.1 Base Tag

See *Tag, Base*.

8.2.2 Binary Unique Identifier

See *BUID*.

8.2.3 BUID

Short for Binary Unique Identifier. Within Synapse, a BUID is the globally unique (within a *Cortex*) SHA-256 digest of a node's msgpack-encoded *Ndef* .

8.3 C

8.3.1 Callable Function

See *Function, Callable*.

8.3.2 Cell

The Cell is a basic building block of a *Synapse Service*, including the *Cortex*. See *Synapse Architecture* for additional detail.

8.3.3 Column, Embed

In *Optic*, a column in Tabular display mode that displays a **property value from an adjacent or nearby node**.

8.3.4 Column, Path Variable

In *Optic*, a column in Tabular display mode that displays **arbitrary data in a column** by defining the data as a *Variable* (a path variable or "path var") within a Storm query.

8.3.5 Column, Property

In *Optic*, a column in Tabular display mode that displays a **property value** from the specified form.

8.3.6 Column, Tag

In *Optic*, a column in Tabular display mode that displays the **timestamps** associated with the specified tag. (Technically, Optic displays two columns - one for each of the min / max timestamps, if present).

8.3.7 Column, Tag Glob

In *Optic*, a column in Tabular display mode that displays any **tags** that match the specified tag or tag glob pattern.

8.3.8 Comparator

Short for *Comparison Operator*.

8.3.9 Comparison Operator

A symbol or set of symbols used in the Storm language to evaluate *Node* property values against one or more specified values. Comparison operators can be grouped into standard and extended operators.

8.3.10 Comparison Operator, Standard

The set of common operator symbols used to evaluate (compare) values in Storm. Standard comparison operators include equal to (=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

8.3.11 Comparison Operator, Extended

The set of Storm-specific operator symbols or expressions used to evaluate (compare) values in Storm based on custom or Storm-specific criteria. Extended comparison operators include regular expression (~=), time/interval (@=), set membership (*in=), tag (#), and so on.

8.3.12 Composite Form

See *Form, Composite*.

8.3.13 Console Tool

See *Tool, Console*.

8.3.14 Constant

In Storm, a constant is a value that cannot be altered during normal execution, i.e., the value is constant.

Contrast with *Variable*. See also *Runtsafe* and *Non-Runtsafe*.

8.3.15 Constructor

Within Synapse, a constructor is code that defines how a *Property* value of a given *Type* can be constructed to ensure that the value is well-formed for its type. Also known as a *Ctor* for short. Constructors support *Type Normalization* and *Type Enforcement*.

8.3.16 Cortex

A Cortex is a *Synapse Service* that implements Synapse's primary data store (as an individual *Hypergraph*). Cortex features include scalability, key/value-based node properties, and a *Data Model* which facilitates normalization.

8.3.17 Cron

Within Synapse, cron jobs are used to create scheduled tasks, similar to the Linux/Unix "cron" utility. The task to be executed by the cron job is specified using the *Storm* query language.

See the Storm command reference for the *cron* command and the *Storm Reference - Automation* document for additional detail.

8.3.18 Ctor

Pronounced "see-tore". Short for *Constructor*.

8.4 D

8.4.1 Daemon

Similar to a traditional Linux or Unix daemon, a Synapse daemon ("dmon") is a long-running or recurring query or process that runs continuously in the background. A dmon is typically implemented by a Storm *Service* and may be used for tasks such as processing elements from a *Queue*. A dmon allows for non-blocking background processing of non-critical tasks. Dmons are persistent and will restart if they exit.

8.4.2 Data Emitter Function

See *Function, Data Emitter*.

8.4.3 Data Model

See *Model, Data*.

8.4.4 Data Model Explorer

In *Optic*, the Data Model Explorer (found in the *Help Tool*) documents and cross-references the current forms and lightweight edges in the Synapse *Data Model*.

8.4.5 Deconflictable

Within Synapse, a term typically used with respect to *Node* creation. A node is deconflictable if, upon node creation, Synapse can determine whether the node already exists within a Cortex (i.e., the node creation attempt is deconflicted against existing nodes). For example, on attempting to create the node `inet:fqdn=woot.com` Synapse can deconflict the node by checking whether a node of the same form with the same primary property already exists.

Most primary properties are sufficiently unique to be readily deconflictable. GUID forms (see *Form, GUID*) require additional considerations for deconfliction. See the *guid* section of the *Storm Reference - Type-Specific Storm Behavior* document for additional detail.

8.4.6 Depadd

Short for "dependent addition". Within Synapse, when a node's secondary property is set, if that secondary property is of a type that is also a form, Synapse will automatically create the node with the corresponding primary property value if it does not already exist. (You can look at this as the secondary property value being "dependent on" the existence of the node with the corresponding primary property value.)

For example, creating the node `inet:email=alice@mail.somecompany.org` will set (via *Autoadd*) the secondary property `inet:email:domain=mail.somecompany.org`. Synapse will automatically create the node `inet:fqdn=mail.somecompany.org` as a dependent addition if it does not exist.

(Note that limited recursion will occur between dependent additions (depadds) and automatic additions (autoadds). When `inet:fqdn=mail.somecompany.org` is created via depadd, Synapse will set (via autoadd) `inet:fqdn:domain=somecompany.org`, which will result in the creation (via depadd) of the node `inet:fqdn=somecompany.org` if it does not exist, etc.)

See also the related concept *Autoadd*.

8.4.7 Derived Property

See *Property, Derived*.

8.4.8 Directed Edge

See *Edge, Directed*.

8.4.9 Directed Graph

See *Graph, Directed*.

8.4.10 Display Mode

In *Optic*, a means of visualizing data using the *Research Tool*. Optic supports the following display modes:

- **Tabular mode,** which displays data and tags in tables (rows of results with configurable columns).
- **Force Graph mode,** which projects data into a directed graph-like view of nodes and their interconnections.
- **Statistics (stats) mode,** which automatically summarizes data using histogram (bar) and sunburst charts.
- **Geospatial mode,** which can be used to plot geolocation data on a map projection.
- **Tree Graph mode,** which displays nodes as a series of vertical "cards" and their property-based links to other nodes.
- **Timeline mode,** which displays nodes with a time property in time sequence order.

8.4.11 Dmon

Short for *Daemon*.

8.5 E

8.5.1 Easy Permissions

In Synapse, easy permissions ("easy perms" for short) are a simplified means to grant common sets of permissions for a particular object to users or roles. Easy perms specify four levels of access, each with a corresponding integer value:

- Deny = 0
- Read = 1
- Edit = 2
- Admin = 3

As an example, the *$lib.macro.grant(name, scope, iden, level)* Storm library can be used to assign easy perms to a *Macro*. Contrast with *Permission*.

8.5.2 Edge

In a traditional *Graph*, an edge is used to connect exactly two nodes (vertexes). Compare with *Hyperedge*.

8.5.3 Edge, Directed

In a *Directed Graph*, a directed edge is used to connect exactly two nodes (vertexes) in a one-way (directional) relationship. Compare with *Hyperedge*.

8.5.4 Edge, Lightweight (Light)

In Synapse, a lightweight (light) edge is a mechanism that links two arbitrary forms via a user-defined verb that describes the linking relationship. Light edges are not forms and so do not support secondary properties or tags. They are meant to simplify performance, representation of data, and Synapse hypergraph navigation for many use cases. Contrast with *Form, Edge*.

8.5.5 Embed Column

See *Column, Embed*.

8.5.6 Entity Resolution

Entity resolution is the process of determining whether different records or sets of data refer to the same real-world entity.

A number of data model elements in Synapse are designed to support entity resolution. For example:

- A `ps:contact` node can capture "a set of observed contact data" for a person (`ps:person`) or organization (`ou:org`). You can link sets of contact data that you assess represent "the same" entity via their `ps:contact:person` or `ps:contact:org` properties.
- A `risk:threat` node can capture "a set of reported data about a threat". If you assess that multiple sources are reporting on "the same" threat, you can link them to an authoritative threat organization via their `risk:threat:org` property.
- An `ou:industryname` node can capture a term used to refer to a commercial industry. You can link variations of a name (e.g., "finance", "financial", "financial services", "banking and finance") to a single `ou:industry` via the `ou:industry:name` and `ou:industry:names` properties.

8.5.7 Extended Comparison Operator

See *Comparison Operator, Extended*.

8.5.8 Extended Form

See *Form, Extended*.

8.5.9 Extended Property

See *Property, Extended*.

8.6 F

8.6.1 Feed

A feed is an ingest API consisting of a set of ingest formats (e.g., file formats, record formats) used to parse records directly into nodes. Feeds are typically used for bulk node creation, such as ingesting data from an external source or system.

8.6.2 Filter

Within Synapse, one of the primary methods for interacting with data in a *Cortex*. A filter operation downselects a subset of nodes from a set of results. Compare with *Lift*, *Pivot*, and *Traverse*.

See *Storm Reference - Filtering* for additional detail.

8.6.3 Filter, Subquery

Within Synapse, a subquery filter is a filter that consists of a *Storm* expression.

See *Subquery Filters* for additional detail.

8.6.4 Fork

Within Synapse, **fork** may refer to the process of forking a *View*, or to the forked view itself.

When you fork a view, you create a new, empty, writable *Layer* on top of the fork's original view. The writable layer from the original view becomes read-only with respect to the fork. Any changes made within a forked view are made within the new writable layer. These changes can optionally be merged back into the original view (in whole or in part), or discarded. (Note that any view-specific automation, such as triggers, dmons, or cron jobs, are **not** copied to the forked view. However, depending on the automation, it may be activated if / when data is merged down into the original view.

8.6.5 Form

A form is the definition of an object in the Synapse data model. A form acts as a "template" that specifies how to create an object (*Node*) within a Cortex. A form consists of (at minimum) a *Primary Property* and its associated *Type*. Depending on the form, it may also have various secondary properties with associated types.

See the *Form* section in the *Data Model Objects* document for additional detail.

8.6.6 Form, Composite

A category of form whose primary property is an ordered set of two or more comma-separated typed values. Examples include DNS A records (`inet:dns:a`) and web-based accounts (`inet:web:acct`).

See also *Form, Edge*.

8.6.7 Form, Edge

A specialized **composite form** (*Form, Composite*) whose primary property consists of two *Ndef* values. Edge forms can be used to link two arbitrary forms via a generic relationship where additional information needs to be captured about that relationship (i.e., via secondary properties and/or tags). Contrast with *Edge, Lightweight (Light)*.

8.6.8 Form, Extended

A custom form added outside of the base Synapse *Data Model* to represent specialized data. Extended forms can be added with the *$lib.model.ext* libraries. **Note** that whenever possible, it is preferable to expand the base Synapse data model to account for novel use cases instead of creating specialized extended forms.

8.6.9 Form, GUID

In the Synpase *Data Model*, a specialized case of a *Simple Form* whose primary property is a *GUID*. The GUID can be either arbitrary or constructed from a specified set of values. GUID forms have additional considerations as to whether or not they are *Deconflictable* in Synapse. Examples of GUID forms include file execution data (e.g., `inet:file:exec:read`) or articles (`media:news`).

8.6.10 Form, Simple

In the Synapse *Data Model*, a category of form whose primary property is a single typed value. Examples include domains (`inet:fqdn`) or hashes (e.g., `hash:md5`).

8.6.11 Function, Callable

In Storm, a callable function is a "regular" function that is invoked (called) and returns exactly one value. A callable function must include a `return()` statement and must not include the `emit` keyword.

8.6.12 Function, Data Emitter

In Storm, a data emitter function emits data. The function returns a generator object that can be iterated over. A data emitter function must include the `emit` keyword and must not include a `return()` statement.

8.6.13 Function, Node Yielder

In Storm, a node yielder function yields nodes. The function returns a generator object that can be iterated over. A node yielder function must not include either the `emit` keyword or a `return()` statement.

8.6.14 Fused Knowledge

See *Knowledge, Fused*.

8.7 G

8.7.1 Gate

See *Auth Gate*.

8.7.2 Global Default Workspace

See *Workspace, Global Default*.

8.7.3 Globally Unique Identifier

See *GUID*.

8.7.4 Graph

A graph is a mathematical structure used to model pairwise relations between objects. Graphs consist of vertices (or nodes) that represent objects and edges that connect exactly two vertices in some type of relationship. Nodes and edges in a graph are typically represented by dots or circles connected by lines.

See *Graphs and Hypergraphs* for additional detail on graphs and hypergraphs.

8.7.5 Graph, Directed

A directed graph is a *Graph* where the edges representing relationships between nodes have a "direction". Given node X and node Y connected by edge E, the relationship is valid for X -> E -> Y but not Y -> E -> X. For example, the relationship "Fred owns bank account #01234567" is valid, but "bank account #01234567 owns Fred" is not. Nodes and edges in a directed graph are typically represented by dots or circles connected by arrows.

See *Graphs and Hypergraphs* for additional detail on graphs and hypergraphs.

8.7.6 GUID

Short for Globally Unique Identifier. Within Synapse, a GUID is a *Type* specified as a 128-bit value that is unique within a given *Cortex*. GUIDs are used as primary properties for forms that cannot be uniquely represented by a specific value or set of values.

8.7.7 GUID Form

See *Form, GUID*.

8.8 H

8.8.1 Help Tool

See *Tool, Help*.

8.8.2 Hive

The Hive is a key/value storage mechanism which is used to persist various data structures required for operating a Synapse *Cell*.

8.8.3 Hyperedge

A hyperedge is an edge within a *Hypergraph* that can join any number of nodes (vs. a *Graph* or *Directed Graph* where an edge joins exactly two nodes). A hyperedge joining an arbitrary number of nodes can be difficult to visualize in flat, two-dimensional space; for this reason hyperedges are often represented as a line or "boundary" encircling a set of nodes, thus "joining" those nodes into a related group.

See *Graphs and Hypergraphs* for additional detail on graphs and hypergraphs.

8.8.4 Hypergraph

A hypergraph is a generalization of a *Graph* in which an edge can join any number of nodes. If a *Directed Graph* where edges join exactly two nodes is two-dimensional, then a hypergraph where a *Hyperedge* can join any number (n-number) of nodes is n-dimensional.

See *Graphs and Hypergraphs* for additional detail on graphs and hypergraphs.

8.9 I

8.9.1 Iden

Short for *Identifier*. Within Synapse, the hexadecimal representation of a unique identifier (e.g., for a node, a task, a trigger, etc.) The term "identifier" / "iden" is used regardless of how the specific identifier is generated.

8.9.2 Identifier

See *Iden*.

8.9.3 Ingest Tool

See *Tool, Ingest*.

8.9.4 Interface

In Synapse, an interface is a data model element that defines a set of secondary properties that are common to a subset of related forms. Forms that should have the set of secondary properties can be defined so as to "inherit" the interface and its properties, as opposed to explicitly declaring each property on every form.

8.9.5 Instance Knowledge

See *Knowledge, Instance*.

8.10 K

8.10.1 Knowledge, Fused

If a form within the Synapse data model has a "range" of time elements (i.e., an interval such as "first seen"/"last seen"), the form typically represents **fused knowledge** – a period of time during which an object, relationship, or event was known to exist. Forms representing fused knowledge can be thought of as combining *n* number of instance knowledge observations. `inet:dns:query`, `inet:dns:a`, and `inet:whois:email` forms are examples of fused knowledge.

See *Instance Knowledge vs. Fused Knowledge* for a more detailed discussion.

8.10.2 Knowledge, Instance

If a form within the Synapse data model has a specific time element (i.e., a single date/time value), the form typically represents **instance knowledge** – a single instance or occurrence of an object, relationship, or event. `inet:dns:request` and `inet:whois:rec` forms are examples of instance knowledge.

See *Instance Knowledge vs. Fused Knowledge* for a more detailed discussion.

8.11 L

8.11.1 Layer

Within Synapse, a layer is the substrate that contains node data and where permissions enforcement occurs. Viewed another way, a layer is a storage and write permission boundary.

By default, a *Cortex* has a single layer and a single *View*, meaning that by default all nodes are stored in one layer and all changes are written to that layer. However, multiple layers can be created for various purposes such as:

- separating data from different data sources (e.g., a read-only layer consisting of third-party data and associated tags can be created underneath a "working" layer, so that the third-party data is visible but cannot be modified);

- providing users with a personal "scratch space" where they can make changes in their layer without affecting the underlying main Cortex layer; or
- segregating data sets that should be visible/accessible to some users but not others.

Layers are closely related to views (see *View*). The order in which layers are instantiated within a view matters; in a multi-layer view, typically only the topmost layer is writeable by that view's users, with subsequent (lower) layers read-only. Explicit actions can push upper-layer writes downward (merge) into lower layers.

8.11.2 Leaf Tag

See *Tag, Leaf*.

8.11.3 Lift

Within Synapse, one of the primary methods for interacting with data in a *Cortex*. A lift is a read operation that selects a set of nodes from the Cortex. Compare with *Pivot*, *Filter*, and *Traverse*.

See *Storm Reference - Lifting* for additional detail.

8.11.4 Lightweight (Light) Edge

See *Edge, Lightweight (Light)*.

8.12 M

8.12.1 Macro

A macro is a stored Storm query. Macros support the full range of Storm syntax and features.

See the Storm command reference for the *macro* command and the *Storm Reference - Automation* for additional detail.

8.12.2 Merge

Within Synapse, merge refers to the process of copying changes made within a forked (see *Fork*) *View* into the original view.

8.12.3 Model

Within Synapse, a system or systems used to represent data and/or assertions in a structured manner. A well-designed model allows efficient and meaningful exploration of the data to identify both known and potentially arbitrary or discoverable relationships.

8.12.4 Model, Analytical

Within Synapse, the set of tags (*Tag*) representing analytical assessments or assertions that can be applied to objects in a *Cortex*.

8.12.5 Model, Data

Within Synapse, the set of forms (*Form*) that define the objects that can be represented in a *Cortex*.

8.13 N

8.13.1 Ndef

Pronounced "en-deff". Short for **node definition.** A node's *Form* and associated value (i.e., *<form>* = *<valu>*) represented as comma-separated elements enclosed in parentheses: `(<form>,<valu>)`.

8.13.2 Node

A node is a unique object within a *Cortex*. Where a *Form* is a template that defines the charateristics of a given object, a node is a specific instance of that type of object. For example, `inet:fqdn` is a form; `inet:fqdn=woot.com` is a node.

See *Node* in the *Data Model Objects* document for additional detail.

8.13.3 Node Action

In *Optic*, a saved, named Storm query or command (action) that can be executed via a right-click context menu option for specified forms (nodes).

8.13.4 Node Data

Node data is a named set of structured metadata that may optionally be stored on a node in Synapse. Node data may be used for a variety of purposes. For example, a *Power-Up* may use node data to cache results returned by a third-party API along with the timestamp when the data was retrieved. If the same API is queried again for the same node within a specific time period, the Power-Up can use the cached node data instead of re-querying the API (helping to prevent using up any API query limits by re-querying the same data).

Node data can be accessed using the node:data type.

8.13.5 Node Definition

See *Ndef*.

8.13.6 Node, Runt

Short for "runtime node". A runt node is a node that does not persist within a Cortex but is created at runtime when a Cortex is initiated. Runt nodes are commonly used to represent metadata associated with Synapse, such as data model elements like forms (`syn:form`) and properties (`syn:prop`) or automation elements like triggers (`syn:trigger`) or cron jobs (`syn:cron`).

8.13.7 Node, Storage

A storage node ("sode") is a collection of data for a given node (i.e., the node's primary property, secondary / universal properties, tags, etc.) that is present in a specific *Layer*.

8.13.8 Node Yielder Function

See *Function, Node Yielder*.

8.13.9 Non-Runtime Safe

See *Non-Runtsafe*.

8.13.10 Non-Runtsafe

Short for "non-runtime safe". Non-runtsafe refers to the use of variables within Storm. A variable that is **non-runtsafe** has a value that may change based on the specific node passing through the Storm pipeline. A variable whose value is set to a node property, such as `$fqdn = :fqdn` is an example of a non-runtsafe variable (i.e., the value of the secondary property `:fqdn` may be different for different nodes, so the value of the variable will be different based on the specific node being operated on).

Contrast with *Runtsafe*.

8.14 O

8.14.1 Optic

The Synapse user interface (UI), available as part of the commercial Synapse offering.

8.15 P

8.15.1 Package

A package is a set of commands and library code used to implement a *Storm Service*. When a new Storm service is loaded into a Cortex, the Cortex verifies that the service is legitimate and then requests the service's packages in order to load any extended Storm commands associated with the service and any library code used to implement the service.

8.15.2 Path Variable Column

See *Column, Path Variable*.

8.15.3 Permission

Within Synapse, a permission is a string (such as `node.add`) used to control access. A permission is assigned (granted or revoked) using a *Rule*.

Access to some objects in Synapse may be controlled by *Easy Permissions*.

8.15.4 Pivot

Within Synapse, one of the primary methods for interacting with data in a *Cortex*. A pivot moves from a set of nodes with one or more properties with specified value(s) to a set of nodes with a property having the same value(s). Compare with *Lift*, *Filter*, and *Traverse*.

See *Storm Reference - Pivoting* for additional detail.

8.15.5 Power-Up

Power-Ups provide specific add-on capabilities to Synapse. For example, Power-Ups may provide connectivity to external databases or third-party data sources, or enable functionality such as the ability to manage YARA rules, scans, and matches.

The term Power-Up is most commonly used to refer to Vertex-developed packages and services that are available as part of the commercial Synapse offering (only a few Power-Ups are available with open-source Synapse). However, many organizations write their own custom packages and services that may also be referred to as Power-Ups.

Vertex distinguishes between an *Advanced Power-Up* and a *Rapid Power-Up*.

8.15.6 Power-Up, Advanced

Advanced Power-Ups are implemented as Storm services (see *Service, Storm*). Vertex-developed Advanced Power-Ups are implemented as Docker containers and may require DevOps support and additional resources to deploy.

8.15.7 Power-Up, Rapid

Rapid Power-Ups are implemented as Storm packages (see *Package*). Rapid Power-Ups are written entirely in Storm and can be loaded directly into a *Cortex*.

8.15.8 Power-Ups Tool

See *Tool, Power-Ups*.

8.15.9 Primary Property

See *Property, Primary*.

8.15.10 Property

Within Synapse, properties are individual elements that define a *Form* or (along with their specific values) that comprise a *Node*. Every property in Synapse must have a defined *Type*.

See the *Property* section in the *Data Model Objects* document for additional detail.

8.15.11 Property Column

See *Column, Property*.

8.15.12 Property, Derived

Within Synapse, a derived property is a secondary property that can be extracted (derived) from a node's primary property. For example, the domain `inet:fqdn=www.google.com` can be used to derive `inet:fqdn:domain=google.com` and `inet:fqdn:host=www`; the DNS A record `inet:dns:a=(woot.com, 1.2.3.4)` can be used to derive `inet:dns:a:fqdn=woot.com` and `inet:dns:a:ipv4=1.2.3.4`.

Synapse will automatically set (*Autoadd*) any secondary properties that can be derived from a node's primary property. Because derived properties are based on primary property values, derived secondary properties are always read-only (i.e., cannot be modified once set).

8.15.13 Property, Extended

Within Synapse, an extended property is a custom property added to an existing form to capture specialized data. For example, extended properties may be added to the data model by a *Power-Up* in order to record vendor-specific data (such as a "risk" score).

Extended properties can be added with the *$lib.model.ext* libraries. **Note** that we strongly recommend that any extended properties be added within a custom namespace; specifically, that property names begin with an underscore and include a vendor or source name (if appropriate) as the first namespace element.

An example of an extended property is the `:_virustotal:reputation` score added to some forms to account for VirusTotal-specific data returned by that Power-Up (e.g., `inet:fqdn:_virustotal:reputation`).

8.15.14 Property, Primary

Within Synapse, a primary property is the property that defines a given *Form* in the data model. The primary property of a form must be defined such that the value of that property is unique across all possible instances of that form. Primary properties are always read-only (i.e., cannot be modified once set).

8.15.15 Property, Relative

Within Synapse, a relative property is a *Secondary Property* referenced using only the portion of the property's namespace that is relative to the form's *Primary Property*. For example, `inet:dns:a:fqdn` is the full name of the "domain" secondary property of a DNS A record form (`inet:dns:a`). `:fqdn` is the relative property / relative property name for that same property.

8.15.16 Property, Secondary

Within Synapse, secondary properties are optional properties that provide additional detail about a *Form*. Within the data model, secondary properties may be defined with optional constraints, such as:

- Whether the property is read-only once set.
- Any normalization (outside of type-specific normalization) that should occur for the property (such as converting a string to all lowercase).

8.15.17 Property, Universal

Within Synapse, a universal property is a *Secondary Property* that is applicable to all forms and may optionally be set for any form where the property is applicable. For example, `.created` is a universal property whose value is the date/time when the associated node was created in a Cortex.

8.16 Q

8.16.1 Queue

Within Synapse, a queue is a basic first-in, first-out (FIFO) data structure used to store and serve objects in a classic pub/sub (publish/subscribe) manner. Any primitive (such as a node iden) can be placed into a queue and then consumed from it. Queues can be used (for example) to support out-of-band processing by allowing non-critical tasks to be executed in the background. Queues are persistent; i.e., if a Cortex is restarted, the queue and any objects in the queue are retained.

8.17 R

8.17.1 Rapid Power-Up

See *Power-Up, Rapid*.

8.17.2 Relative Property

See *Property, Relative*.

8.17.3 Repr

Short for "representation". The repr of a *Property* defines how the property should be displayed in cases where the display format differs from the storage format. For example, date/time values in Synapse are stored in epoch milliseconds but are displayed in human-friendly "yyyy/mm/dd hh:mm:ss.mmm" format.

8.17.4 Research Tool

See *Tool, Research*.

8.17.5 Role

In Synapse, a role is used to group users with similar authorization needs. You can assign a set of rules (see *Rule*) to a role, and grant the role to users who need to perform those actions.

8.17.6 Root Tag

See *Tag, Root*.

8.17.7 Rule

Within Synapse, a rule is a structure used to assign (grant or prohibit) a specific *Permission* (e.g., `node.tag` or `!view.del`). A rule is assigned to a *User* or a *Role*.

8.17.8 Runt Node

See *Node, Runt*.

8.17.9 Runtime Safe

See *Runtsafe*.

8.17.10 Runtsafe

Short for "runtime safe". Runtsafe refers to the use of variables within Storm. A variable that is **runtsafe** has a value that will not change based on the specific node passing through the Storm pipeline. A variable whose value is explicitly set, such as `$fqdn = woot.com` is an example of a runtsafe varaible.

Contrast with *Non-Runtsafe*.

8.18 S

8.18.1 Secondary Property

See *Property, Secondary*.

8.18.2 Service

Synapse is designed as a modular set of services. Broadly speaking, a service can be thought of as a container used to run an application. We may informally differentiate between a *Synapse Service* and a *Storm Service*.

8.18.3 Service, Storm

A Storm service is a registerable remote component that can provide packages (*Package*) and additional APIs to Storm and Storm commands. A service resides on a *Telepath* API endpoint outside of the *Cortex*.

When the Cortex is connected to a service, the Cortex queries the endpoint to determine if the service is legitimate and, if so, loads the associated package to implement the service.

An advantage of Storm services (over, say, additional Python modules) is that services can be restarted to reload their service definitions and packages while a Cortex is still running – thus allowing a service to be updated without having to restart the entire Cortex.

8.18.4 Service, Synapse

Synapse services make up the core Synapse architecture and include the *Cortex* (data store), *Axon* (file storage), and the commercial *Optic* UI. Synapse services are built on the *Cell* object.

8.18.5 Simple Form

See *Form, Simple*.

8.18.6 Slab

A Slab is a core Synapse component which is used for persisting data on disk into a LMDB backed database. The Slab interface offers an asyncio friendly interface to LMDB objects, while allowing users to largely avoid having to handle native transactions themselves.

8.18.7 Sode

Short for “storage node”. See *Node, Storage*.

8.18.8 Splice

A splice is an atomic change made to data within a Cortex, such as node creation or deletion, adding or removing a tag, or setting, modifying, or removing a property. All changes within a Cortex may be retrieved as individual splices within the Cortex's splice log.

8.18.9 Spotlight Tool

See *Tool, Spotlight*.

8.18.10 Standard Comparison Operator

See *Comparison Operator, Standard*.

8.18.11 Storage Node

See *Node, Storage*.

8.18.12 Stories Tool

See *Tool, Stories*.

8.18.13 Storm

Storm is the custom query language analysts use to interact with data in Synapse.

Storm can also be used as a programming language by advanced users and developers, though this level of expertise is not required for normal use. Many of Synapse's **Power-Ups** (see *Power-Up*) are written in Storm.

See *Storm Reference - Introduction* for additional detail.

8.18.14 Storm Editor

Also "Storm Editor Tool". See *Tool, Storm Editor*.

8.18.15 Storm Service

See *Service, Storm*.

8.18.16 Subquery

Within Synapse, a subquery is a *Storm* query that is executed inside of another Storm query.

See *Storm Reference - Subqueries* for additional detail.

8.18.17 Subquery Filter

See *Filter, Subquery*.

8.18.18 Synapse Service

See *Service, Synapse*.

8.19 T

8.19.1 Tag

Within Synapse, a tag is a label applied to a node that provides additional context about the node. Tags typically represent assessments or judgements about the data represented by the node.

See the *Tag* section in the *Data Model Objects* document for additional detail.

8.19.2 Tag, Base

Within Synapse, the lowest (rightmost) tag element in a tag hierarchy. For example, for the tag `#foo.bar.baz`, `baz` is the base tag.

8.19.3 Tag, Leaf

The full tag path / longest tag in a given tag hierarchy. For example, for the tag `#foo.bar.baz`, `foo.bar.baz` is the leaf tag.

8.19.4 Tag, Root

Within Synapse, the highest (leftmost) tag element in a tag hierarchy. For example, for the tag `#foo.bar.baz`, `foo` is the root tag.

8.19.5 Tag Column

See *Column, Tag*.

8.19.6 Tag Explorer

In *Optic*, the Tag Explorer (found in the *Help Tool*) provides an expandable, tree-based listing of all tags in your Synapse *Cortex*, along with their definitions (if present).

8.19.7 Tag Glob Column

See *Column, Tag Glob*.

8.19.8 Taxonomy

In Synapse, a taxonomy is a user-defined set of hierarchical categories that can optionally be used to further classify particular objects (forms). Taxonomies use a dotted namespace (similar to tags). Forms that support a taxonomy will have a secondary property whose *Type* is the taxonomy for that form (e.g., an `ou:industry` form has a `:type` secondary property whose type is `ou:industry:type:taxonomy`).

8.19.9 Telepath

Telepath is a lightweight remote procedure call (RPC) protocol used in Synapse. See *Telepath RPC* in the *Synapse Architecture* guide for additional detail.

8.19.10 Tool, Admin

In *Optic*, the Admin Tool provides a unified interface to perform basic management of users, roles, and permissions; views and layers; and triggers and cron jobs.

8.19.11 Tool, Console

In *Optic*, the Console Tool provides a CLI-like interface to Synapse. It can be used to run Storm queries in a manner similar to the Storm CLI (in the community version of Synapse). In Optic the Console Tool is more commonly used to display status, error, warning, and debug messages, or to view help for built-in Storm commands (see *Storm Reference - Storm Commands*) and / or Storm commands installed by Power-Ups.

8.19.12 Tool, Help

In *Optic*, the central repository for Synapse documentation and assistance. The Help Tool includes the *Data Model Explorer*, *Tag Explorer*, documentation for any installed Power-Ups (see *Power-Up*), links to the public Synapse, Storm, and Optic documents, and version / changelog information.

8.19.13 Tool, Ingest

In *Optic*, the primary tool used to load structured data in CSV, JSON, or JSONL format into Synapse using Storm. The Ingest Tool can also be used to prototype and test more formal ingest code.

8.19.14 Tool, Power-Ups

In *Optic*, the tool used to view, install, update, and remove Power-Ups (see *Power-Up*).

8.19.15 Tool, Research

In *Optic*, the primary tool used to ingest, enrich, explore, visualize, and annotate Synapse data.

8.19.16 Tool, Spotlight

Also known as simply "Spotlight". In *Optic*, a tool used to load and display PDF or HTML content, create an associated `media:news` node, and easily extract and link relevant indicators or other nodes.

8.19.17 Tool, Stories

Also known as simply "Stories". In *Optic*, a tool used to create, collaborate on, review, and publish finished reports. Stories allows you to integrate data directly from the *Research Tool* into your report ("Story").

8.19.18 Tool, Storm Editor

Also known as simply "Storm Editor". In *Optic*, a tool used to compose, test, and store Storm queries (including macros - see *Macro*). Storm Editor includes a number of integrated development environment (IDE) features, including syntax highlighting, auto-indenting, and auto-completion for the names of forms, properties, tags, and libraries.

8.19.19 Tool, Workflows

In *Optic*, the tool used to access and work with Workflows (see *Workflow*).

8.19.20 Tool, Workspaces

In *Optic*, the tool used to configure and manage a user's Workspaces (see *Workspace*).

8.19.21 Traverse

Within Synapse, one of the primary methods for interacting with data in a *Cortex*. Traversal refers to navigating the data by crossing ("walking") a lighweight (light) edge (*Edge, Lightweight (Light)*) betweeen nodes. Compare with *Lift*, *Pivot*, and *Filter*.

See walk-light-edge for additional detail.

8.19.22 Trigger

Within Synapse, a trigger is a Storm query that is executed automatically upon the occurrence of a specified event within a Cortex (such as adding a node or applying a tag). “Trigger” refers collectively to the event and the query fired (“triggered”) by the event.

See the Storm command reference for the *trigger* command and the *Storm Reference - Automation* for additional detail.

8.19.23 Type

Within Synapse, a type is the definition of a data element within the data model. A type describes what the element is and enforces how it should look, including how it should be normalized.

See the *Type* section in the *Data Model Objects* document for additional detail.

8.19.24 Type, Base

Within Synapse, base types include standard types such as integers and strings, as well as common types defined within or specific to Synapse, including globally unique identifiers (`guid`), date/time values (`time`), time intervals (`ival`), and tags (`syn:tag`). Many forms within the Synapse data model are built upon (extensions of) a subset of common types.

8.19.25 Type, Model-Specific

Within Synapse, knowledge-domain-specific forms may themselves be specialized types. For example, an IPv4 address (`inet:ipv4`) is its own specialized type. While an IPv4 address is ultimately stored as an integer, the type has additional constraints, e.g., IPv4 values must fall within the allowable IPv4 address space.

8.19.26 Type Awareness

Type awareness is the feature of the *Storm* query language that facilitates and simplifies navigation through the *Hypergraph* when pivoting across nodes. Storm leverages knowledge of the Synapse *Data Model* (specifically knowledge of the type of each node property) to allow pivoting between primary and secondary properties of the same type across different nodes without the need to explicitly specify the properties involved in the pivot.

8.19.27 Type Enforcement

Within Synapse, the process by which property values are required to conform to value and format constraints defined for that *Type* within the data model before they can be set. Type enforcement helps to limit bad data being entered in to a Cortex by ensuring values entered make sense for the specified data type (e.g., that an IP address cannot be set as the value of a property defined as a domain (`inet:fqdn`) type, and that the integer value of the IP falls within the allowable set of values for IP address space).

8.19.28 Type Normalization

Within Synapse, the process by which properties of a particular type are standardized and formatted in order to ensure consistency in the data model. Normalization may include processes such as converting user-friendly input into a different format for storage (e.g., converting an IP address entered in dotted-decimal notation to an integer), converting certain string-based values to all lowercase, and so on.

8.20 U

8.20.1 Universal Property

See *Property, Universal*.

8.20.2 User

In Synapse, a user is represented by an account in the Cortex. An account is required to authenticate (log in) to the Cortex and is used for authorization (permissions) to access services and perform operations.

8.21 V

8.21.1 Variable

In Storm, a variable is an identifier with a value that can be defined and/or changed during normal execution, i.e., the value is variable.

Contrast with *Constant*. See also *Runtsafe* and *Non-Runtsafe*.

See *Storm Reference - Advanced - Variables* for a more detailed discussion of variables.

8.21.2 Vault

In Synapse, a vault is a protected storage mechanism that allows you to store secret values (such as API keys) and any associated configuration settings. Vaults support permissions and can be shared with other users or roles. Granting 'read' access to a vault allows someone to use the vault contents without allowing them to see the vault's secret values.

8.21.3 View

Within Synapse, a view is a ordered set of layers (see *Layer*) and associated permissions that are used to synthesize nodes from the *Cortex*, determining both the nodes that are visible to users via that view and where (i.e., in what layer) any changes made by a view's users are recorded. A default Cortex consists of a single layer and a single view, meaning that by default all nodes are stored in one layer, all changes are written to that layer, and all users have the same visibility (view) into Synapse's data.

In multi-layer systems, a view consists of the set of layers that should be visible to users of that view, and the order in which the layers should be instantiated for that view. Order matters because typically only the topmost layer is writeable by that view's users, with subsequent (lower) layers read-only. Explicit actions can push upper-layer writes downward (merge) into lower layers.

8.22 W

8.22.1 Workflow

In *Optic*, a Workflow is a customized set of UI elements that provides an intuitive way to perform particular tasks. Workflows may be installed by Synapse Power-Ups (see *Power-Up*) and give users a more tailored means (compared to the *Research Tool* or Storm query bar) to work with Power-Up Storm commands or associated analysis tasks.

8.22.2 Workflows Tool

See *Tool, Workflows*.

8.22.3 Workspace

In *Optic*, a Workspace is a customizable user environment. Users may configure one or more Workspaces; different Workspaces may be designed to support different analysis tasks.

8.22.4 Workspace, Global Default

In *Optic*, a Workspace that has been pre-configured with various custom settings and distributed for use. A Global Default Workspace can be used to share a set of baseline Workspace customizations with a particular group or team.

8.22.5 Workspaces Tool

See *Tool, Workspaces*.

CHAPTER NINE

SYNAPSE CONTRIBUTORS GUIDE

This Contributors Guide is written for people who will be working on the Synapse code base, contributing to it via code patches, or maintaining written documentation.

The Contributors Guide is a living document and will continue to be updated and expanded. The current sections are:

9.1 Contributing to Synapse

- *Project Style Guide*.
- *Git Hook & Syntax Checking*.
- *Contribution Process*.

9.1.1 Project Style Guide

The following items should be considered when contributing to Synapse:

- The project is not currently strictly PEP8 compliant. Compliant sections include the following:
  - Whitespace in Expressions and Statements.
  - Programming Recommendations regarding singleton comparison (use 'is' instead of equality operators).
- Please keep line lengths under 120 characters.
- Use single quotes for string constants (including docstrings) unless double quotes are required.

```
# Do this
foo = '1234'
# NOT this
foo = "1234"
```

- Use a single line break between top level functions and class definitions, and class methods. This helps conserve vertical space.
  - Do this

```
import foo
import duck

def bar():
    return True

```

(continues on next page)

(continued from previous page)

```
def baz():
    return False

class Obj(object):

    def __init__(self, a):
        self.a = a

    def gimmeA(self):
        return self.a
```

∗ NOT this

```
import foo
import duck


def bar():
    return True


def baz():
    return False


class Obj(object):

    def __init__(self, a):
        self.a = a


    def gimmeA(self):
        return self.a
```

- Use Google style Python docstrings. This format is very readable and will allow type hinting for IDE users. See the following notes below about our slight twist on this convention.
  - Use '' quotes instead of """ for starting/stoping doc strings.
  - Google Style typically has the summary line after the opening '' marker. Place this summary value on the new line following the opening '' marker.
  - More information about Google Style docstrings (and examples) can be found at the examples here.
  - We use Napoleon for parsing these doc strings. More info here.
  - Synapse as a project is not written using the Napoleon format currently but all new modules should adhere to that format.
  - Synapse acceptable example:

```
def fooTheBar(param1, param2, **kwargs):
    '''
    Summary line goes first.

```

(continues on next page)

(continued from previous page)

```
    Longer description lives here. It can be a bunch of stuff across
    multiple blocks if necessary.

    Example:
        Examples should be given using either the ``Example`` section.
        Sections support any reStructuredText formatting, including
        literal blocks::

            woah = fooTheBar('a', 'b', duck='quacker')

    Section breaks are created by resuming unindented text. Section breaks
    are also implicitly created anytime a new section starts.

    `PEP 484`_ type annotations are supported. If attribute, parameter, and
    return types are annotated according to `PEP 484`_, they do not need to be
    included in the docstring:

    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.

    Keyword Arguments:
        duck (str): Optional keyword args which come in via **kwargs call␣
↪conventions,
                    which modify function behavior, should be documented under␣
↪the
                    Keyword Args section.

    Returns:
        bool: The return value. True for success, False otherwise.

              The ``Returns`` section supports any reStructuredText formatting,
              including literal blocks::

                  {
                      'param1': param1,
                      'param2': param2
                  }

    Raises:
        AttributeError: The ``Raises`` section is a list of all exceptions
            that are relevant to the interface.
        ValueError: If `param2` is equal to `param1`.


    .. _PEP 484:
        https://www.python.org/dev/peps/pep-0484/

    '''
    # Do stuff the with args...
```

- Imports should first be sorted in order of shortest to longest import, then by alphabetical order (when lengths match). Imports should be ordered starting from the Python standard library first, then any third party packages, then any Synapse specific imports. The following example shows the recommended styling for imports:

```
# Stdlib
import logging
import collections
# Third Party Code
import barlib.duck as b_duck
import foolib.thing as f_thing
# Synapse Code
import synapse.common as s_common
import synapse.cortex as s_cortex
import synapse.lib.config as s_config
```

- Previously we used * imports in the Synapse codebase (especially around synapse.exc and synapse.common). If common functions or exceptions are needed, import synapse.common as noted above, and both the common functions and the entirety of synapse.exc exceptions will be available. This provides a consistent manner for referencing common functions and Synapse specific exception classes. New code should generally not use * imports. Here is an example:

```
# Do this
import synapse.common as s_common
tick = s_common.now()
if tick < 1000000000:
   raise s_common.HitMaxTime(mesg='We have gone too far!')

# NOT this
from synapse.common import *
tick = now()
if tick < 1000000000:
   raise HitMaxTime(mesg='We have gone too far!')
```

- Function names should follow the mixedCase format for anything which is exposed as a externally facing API on a object or module.

```
# Do this
fooTheBar()
# NOT this
foo_the_bar()
```

- Private methods should be marked as such with a proceeding underscore.

```
# Do this
_internalThing()
# NOT this
privateInternalThingDontUseMe()
```

  – The corollary to this is that any function which is not private may be called arbitrarily at any time, so avoid public API functions which are tightly bound to instance state. For example, if a processing routine is broken into smaller subroutines for readability or testability, these routines are likely private and should not be exposed to outside callers.

- Function calls with mandatory arguments should be called with positional arguments. Do not use keyword arguments unless necessary.

```
def foo(a, b, duck=None):
   print(a, b, duck)

# Do this
foo('a', 'b', duck='quacker')
# Not this
foo(a='a', b='b', duck='quacker')
```

- Avoid the use of @property decorators. They do not reliably work over the telepath RMI.
- Logging should be setup on a per-module basis, with loggers created using calls to logging.getLogger(__name__). This allows for module level control of loggers as neccesary.
  - Logger calls should use logging string interpolation, instead of using % or .format() methods. See Python Logging module docs for reference.
  - Example:

    ```
    # Get the module level logger
    logger = logging.getLogger(__name__)
    # Do this - it only forms the final string if the message is
    # actually going to be logged
    logger.info('I am a message from %s about %s', 'bob', 'a duck')
    # NOT this - it performs the string format() call regardless of
    # whether or not the message is going to be logged.
    logger.info('I am a message from {} about {}'.format('bob', 'a duck'))
    ```

- Convenience methods are available for unit tests, primarily through the SynTest class. This is a subclass of unittest.TestCase and provides many short aliases for the assert* functions that TestCase provides.
  - Ensure you are closing resources which may be open with test cases. Many Synapse objects may be used as content managers which make this easy for test authors.
- Avoid the use of the built-in `re` module. Instead use the third-party `regex` module. `regex` is preferred due to known bugs with unicode in the `re` module. Additionally, `regex` does provide some performance benefits over `re`, especially when using pre-compiled regular expression statements.
- Whenever possible, regular expressions should be pre-compiled. String matches/comparisons should be performed against the pre-compiled regex instance.

```
# Do this
fqdnre = regex.compile(r'^[\w._-]+$', regex.U)

def checkValue(valu):
    if not fqdnre.match(valu):
        self._raiseBadValu(valu)

# NOT this
def checkValue(valu):
    if not regex.match(r'^[\w._-]+$', valu, regex.U)
        self._raiseBadValu(valu)
```

- Return values should be preferred over raising exceptions. Functions/methods that return a value should return None (or a default value) in the case of an error. The logic behind this is that it is much easier, cleaner, faster to check a return value than to handle an exception.

  Raising exceptions is reserved for “exceptional circumstances” and should not be used for normal program flow.

```
# Do this
def getWidgetById(self, wid):
    widget_hash = self._index.get(wid)
    if widget_hash is None:
        return None

    widget = self._widgets.get(widget_hash)
    return widget

# NOT this
def getWidgetById(self, wid):
    widget_hash = self._index.get(wid)
    if widget_hash is None:
        raise NotFoundError

    widget = self._widgets.get(widget_hash)
    if widget is None:
        raise NotFoundError

    return widget
```

Contributions to Synapse which do not follow the project style guidelines may not be accepted.

9.1.2 Git Hook & Syntax Checking

A set of helper scripts are available for doing python syntax checking. Basic syntax checking can be run with the `pycodestyle` tool; while a a git pre-commit hook; and a script to run autopep8 on staged git files also exist to make life easier.

The pre-commit hook does syntax checking on .py files which contain invalid syntax. The hook will **ALSO** run nbstripout on .ipynb files to remove output data from cells. This results in cleaner diffs for .ipynb files over time.

1. An example of running the generic syntax check script is seen below:

```
~/git/synapse$ python -m pycodestyle
./synapse/tests/test_lib_types.py:397: [E226] missing whitespace around arithmetic␣
↪operator
./synapse/tests/test_lib_types.py:398: [E226] missing whitespace around arithmetic␣
↪operator
```

2. Installing the git hook is easy:

```
cp scripts/githooks/pre-commit .git/hooks/pre-commit
chmod +x .git/hooks/pre-commit
```

3. After installing the hook, attempting a commit with a syntax error will fail

```
~/git/synapse$ git commit -m "Demo commit"
PEP8 style violations have been detected.  Please fix them
or force the commit with "git commit --no-verify".

./synapse/tests/test_lib_types.py:397: [E226] missing whitespace around arithmetic␣
↪operator
```

(continues on next page)

(continued from previous page)

```
./synapse/tests/test_lib_types.py:398: [E226] missing whitespace around arithmetic␣
↪operator
```

4. This may be automatically fixed for you using the `pep8_staged_files.py` script. Note that **most**, but not **all** syntax errors may be fixed with the helper script.

```
# Run the pep8_staged_files.py script
~/git/synapse$ ./scripts/pep8_staged_files.py
# Check the diff
~/git/synapse$ git diff synapse/tests/test_lib_types.py
diff --git a/synapse/tests/test_lib_types.py b/synapse/tests/test_lib_types.py
index 0e3a7498..b81575ef 100644
--- a/synapse/tests/test_lib_types.py
+++ b/synapse/tests/test_lib_types.py
 class TypesTest(s_t_utils.SynTest):

     def test_type(self):
@@ -397,8 +395,8 @@ class TypesTest(s_t_utils.SynTest):
             self.eq({node.ndef[1] for node in nodes}, {'m'})
             nodes = await alist(core.eval('testcomp +testcomp*range=((1024,␣
↪grinch), (4096, zemeanone))'))
             self.eq({node.ndef[1] for node in nodes}, {(2048, 'horton'), (4096,
↪'whoville')})
-            guid0 = 'B'*32
-            guid1 = 'D'*32
+            guid0 = 'B' * 32
+            guid1 = 'D' * 32
             nodes = await alist(core.eval(f'testguid +testguid*range=({guid0},
↪{guid1})'))
             self.eq({node.ndef[1] for node in nodes}, {'c' * 32})
             nodes = await alist(core.eval('testint | noderefs |␣
↪+testcomp*range=((1000, grinch), (4000, whoville))'))

# Add the file and commit
~/git/synapse$ git add synapse/tests/test_lib_types.py
~/git/synapse$ git commit -m "Demo commit"
[some-branch f254f5bf] Demo commit
 1 file changed, 3 insertions(+), 2 deletions(-)
```

9.1.3 Contribution Process

The Vertex Project welcomes contributions to the Synapse Hypergraph framework in order to continue its growth!

In order to contribute to the project, do the following:

1. Fork the Synapse repository from the Vertex Project. Make a new branch in git with a descriptive name for your change. For example:

```
git checkout -b foohuman_new_widget
```

2. Make your changes. Changes should include the following information:

   - Clear documentation for new features or changed behavior

   - Unit tests for new features or changed behaviors
   - If possible, unit tests should also show minimal use examples of new features.

3. Ensure that both your tests and existing Synapse tests successfully run. You can do that manually via the python unittest module, or you can set up CircleCI to run tests for your fork (this is a exercise for the reader). The following examples shows manual test runs:

   ```
   pytest -v
   pytest -v synapse/tests/your_test_file.py
   ```

   If test coverage is desired, you can use the provided testrunner.sh shell script to run a test. This script will generate HTML coverage reports and attempt to open those reports using xdg-open. This requires the pytest, pytest-cov, pytest-xdist packages to be installed.

   ```
   ./scripts/testrunner.sh
   ./scripts/testrunner.sh synapse/tests/your_test_file.py
   ./scripts/testrunner.sh synapse/tests/your_test_file.py::YourTestClass
   ./scripts/testrunner.sh synapse/tests/your_test_file.py::YourTestClass::test_
   ↪function
   ```

4. Rebase your feature branch on top of the latest master branch of the Vertex Project Synapse repository. This may require you to add the Vertex Project repository to your git remotes. The following example of rebasing can be followed:

   ```
   # Add the Vertex project repository as a remote named "upstream".
   git remote add upstream https://github.com/vertexproject/synapse.git
   # Grab data from the upstream repository
   git fetch --all
   # Change to your local git master branch
   git checkout master
   # Merge changes from upstream/master to your local master
   git merge upstream/master
   # Move back to your feature branch
   git checkout foohuman_new_feature
   # Rebase your feature branch ontop of master.
   # This may require resolving merge conflicts.
   git rebase master
   # Push your branch up to to your fork - this may require a --force
   # flag if you had previously pushed the branch prior to the rebase.
   git push
   ```

5. Ensure your tests still pass with the rebased feature branch.
6. If your changes require extensive documentation, please very your API documentation builds properly and any additional user or devops docs are created as needed. See *Synapse Doc Mastering* for documentation mastering notes.
7. Create the Pull Request in Github, from your fork's feature branch to the master branch of the Vertex Project Synapse repository. Include a description and a reference to any open issues related to the PR.

9.2 Synapse Doc Mastering

Documentation for creation and generation of documentation for Synapse.

9.2.1 Generating Docs Locally

API documentation is automatically generated from docstrings, and additional docs may also be added to Synapse as well for more detailed discussions of Synapse subsystems. This is currently done via readthedocs.

In order to do local doc generation you can do the following steps:

1. Install the following packages (preferably in a virtualenv):

```
# cd to your synapse checkout
cd synapse
# Install additional packages - this assumes the environment already has
# any additional packages required for executing synapse code in it.
python -m pip install -U -r requirements_doc.txt
# Alternativly, you can install synapse directly in develop mode with pip
# python -m pip install .[docs]

# Install pandoc package, required for building HTML.
# This may require sudo access depending on your environment.
apt install pandoc
```

2. Build the docs using sphinx. A makefile is provided which makes this easy.

```
# Go to your synapse repo
cd synapse
# Go to the docs folder
cd docs
# Use the make command to build the HTML docs
make html
```

3. Now you can open the HTML docs for browsing them.

```
xdg-open _build/html/index.html
```

4. To rebuild documentation from scratch you can delete the _build directory and the `api` directories. Deleting the `api` directory will cause the automatic Synapse API documentation to be rebuilt.

```
# Delete the _build directory
make clean
# Remove all old files and remove the autodocs directory
rm -rf synapse/autodocs
```

9.2.2 Mastering Docs

Synapse documents are mastered using either raw ReStructuredText (.rst) files or as Jupyter Notebooks (.ipynb). Notebooks should be used for documenting anything which may include Storm or code examples, so that the examples can be written in a manner that can be asserted, so the documentation can be tested in the CI pipeline. Notebooks are also executed during sphinx document build steps, so any output is current as of document build time. Text in Notebooks should be mastered as RST using raw NbConvert cells.

In general, docs for Synapse fall into two categories: User guides and devops guides. User guides should be mastered in `./docs/synapse/userguides` and devops guides should be mastered in `./docs/synapse/devops`. Additional top level sections may be added over time.

In order to master Notebooks, you will need to setup the hide_code extension for Jupyter. That is used to selectively hide code and output blocks as needed. For example, this allows use to hide the code used to run a Storm command and show the output.

The following steps are a high level overview of the process to setup Jupyter and add or edit notebooks for documentation purposes.

- Setup the hide_code extension:

  ```
  # Then install & enable the Jupyter hide-code extension
  # This only has to be run once.
  jupyter nbextension install --py --user hide_code
  jupyter nbextension enable --py --user hide_code
  jupyter serverextension enable --py --user hide_code
  ```

- Launch Jupyter to run a local notebook server:

  ```
  # Go to your synapse repo
  cd synapse
  # Launch the notebook server
  jupyter notebook
  ```

- Navigate to the docs directory in Jupyter. Create a new notebook or open an existing notebook as needed. This will likely be located under the `docs/synapse/userguides` or `docs/synapse/devops` directories.
- For Storm CLI integration, you can add the following code block into the first code cell in order to get some Synapse Jupyter helpers:

```
import os, sys
try:
    from synapse.lib.jupyter import *
except ImportError as e:
    # Insert the root path of the repository to sys.path.
    # This assumes the notebook is located three directories away
    # From the root synapse directory. It may need to be varied
    synroot = os.path.abspath('../../../')
    sys.path.insert(0, synroot)
    from synapse.lib.jupyter import *
```

- You can use helpers to execute storm commands in the following fashion to get a CoreCmdr object, execute a storm query printing the CLI ouput to screen, while asserting the number of nodes returned, and then closing the object.

```
# Get a CoreCmdr object
corecmdr = await getTempCoreCmdr()
# Execute the query and get the packed nodes.
podes = await corecmdr.eval('[inet:ipv4=1.2.3.4]',
                            num=1, cmdr=True)
```

```
Detected 159 deprecated properties unlocked and not in use, recommend locking (https://v.
↪vtx.lk/deprlock).
```

```
WARNING: "cmdr" is deprecated in 2.164.0 and will be removed in 3.0.0
```

```
cli> storm [inet:ipv4=1.2.3.4]
```

```
Executing query at 2024/04/19 14:25:33.896
```

```
...
```

```
inet:ipv4=1.2.3.4
```

```
.created = 2024/04/19 14:25:36.208
```

```
:type = unicast
```

```
complete. 1 nodes in 2313 ms (0/sec).
```

- We have a helper function available from the `synapse.lib.jupyter` imported earlier called `getDocData(fn)`. It will look for a given filename in the `docs/docdata` directory; and get its data. If the file ends with `.json`, `.jsonl`, `.yaml`, or `.mpk` we will return the decoded data, otherwise we will return the raw bytes. This uses a function called `getDocPath(fn)` which will find and return a file under the `docs\docdata` directory.

  There is an example below showing the use of this to load a json file located at `docs/docdata/mastering_example_ingest.json`, and adding the data to the Cortex via the `addFeedData()` function.

```
fn = 'mastering_example_ingest.json'
data = getDocData(fn)
await corecmdr.addFeedData('syn.nodes', data)
podes = await corecmdr.eval('#example', num=2, cmdr=True)
```

```
cli> storm #example
```

```
Executing query at 2024/04/19 14:25:36.257
```

```
inet:ipv4=0.0.0.1
```

```
.created = 2024/04/19 14:25:36.238
```

```
:type = private
```

```
#example
```

```
inet:fqdn=woot.com
```

```
.created = 2024/04/19 14:25:36.239
```

```
:domain = com
```

```
:host = woot
```

```
:issuffix = false
```

```
:iszone = true
```

```
:zone = woot.com
```

```
#example
```

```
complete. 2 nodes in 17 ms (117/sec).
```

- Since the Code cells are persistent, you can reuse the objects from earlier cells until a resource has been closed (`.fini()`'d). The following example shows using the `corecmdr` object from the above code section to lift a node and print it to the screen.

```
from pprint import pprint # We want to make our nodes pretty
podes = await(corecmdr.eval('inet:ipv4'))
for pode in podes:
    pprint(pode)
```

```
(
```

```
('inet:ipv4', 1)
```

```
,
```

```
{
```

```
'iden'
```

```
:
```

```
'2f70f448adcc6e9b9846aecfd034efc4f9d583e614f1b3489d1cf1d32fb64667'
```

```
,
```

```
'nodedata'
```

```
:
```

```
{}
```

```
,
```

```
'path'
```

```
:
```

```
{}
```

```
,
```

```
'props'
```

```
:
```

```
{'.created': 1713536736238, 'type': 'private'}
```

```
,
```

```
'tagprops'
```

```
:
```

```
{}
```

```
,
```

```
'tags'
```

```
:
```

```
{'example': (None, None)}
```

```
}
```

```
)
```

```
(
```

```
('inet:ipv4', 16909060)
```

```
,
```

```
{
```

```
'iden'
```

```
:
```

```
'20153b758f9d5eaaa38e4f4a65c36da797c3e59e549620fa7c4895e1a920991f'
```

```
,
```

```
'nodedata'
```

```
:
```

```
{}
```

```
,
```

```
'path'
```

```
:
```

```
{}
```

```
,
```

```
'props'
```

```
:
```

```
{'.created': 1713536736208, 'type': 'unicast'}
```

```
,
```

```
'tagprops'
```

```
:
```

```
{}
```

```
,
```

```
'tags'
```

```
:
```

```
{}
```

```
}
```

```
)
```

- We can also execute a line of text in the CLI directly with the `runCmdLine()` function. For example, we can use this to execute the `help` command and see all available commands to the raw CLI object. This will always print the CLI output to the Jupyter cell output.

```
# Run the help command.
text = 'help'
await corecmdr.runCmdLine(text)
```

```
cli> help
```

```
help  - List commands and display help output.
```

```
hive  - Manipulates values in a cell's Hive.
```

```
kill  - Kill a running task/query within the cortex.
```

```
locs  - List the current locals for a given CLI object.
```

```
log   - Add a storm log to the local command session.
```

```
ps    - List running tasks in the cortex.
```

```
quit  - Quit the current command line interpreter.
```

```
storm - Execute a storm query.
```

- In the above example, there is some Python syntax highlighting occuring. This may not be desired. In order to disable that, add the following to the first line of the RST body of a document:

  ```
  .. highlight::  none
  ```

  This will disable all code highlighting in a given document, until another `highlight` directive is encountered.

- The following code and output will have their highlighting disabled, via the use of a pair of `highlight` directives before and after the code cell. The first directive disabled highlighting, and the subsequent directive re-enabled it for python3 highlighting.

  Read the Sphinx Literal documentation for additional information about highlighting controls.

```
# Run the help command again.
text = 'help'
await corecmdr.runCmdLine(text)
```

```
cli> help
```

```
help  - List commands and display help output.
```

```
hive  - Manipulates values in a cell's Hive.
```

```
kill  - Kill a running task/query within the cortex.
```

```
locs  - List the current locals for a given CLI object.
```

```
log   - Add a storm log to the local command session.
```

```
ps    - List running tasks in the cortex.
```

```
quit  - Quit the current command line interpreter.
```

```
storm - Execute a storm query.
```

- When we are done with the CoreCmdr object, we should `fini()` is to remove any resources it may have created. This is done below.

```
# Close the object.
_ = await corecmdr.fini()
```

- You can enable the hide_code options by selecting the "View -> Cell Toolbar -> Hide code" option. This will allow you to optionally hide code or output blocks.
- After adding text and code to a notebook, ensure that it runs properly and any produces the expected outputs. You can then mark any code cells for hiding as necessary; then save your notebook. You can then follow the earlier instructions for how to build and view the docs locally.
- Once new documents are made, they will needto be added to the appropriate toctree directive. There are three index documents:
  - index.rst - This controls top-level documentation ordering. It generally should not need to be edited unless adding a new top level document or adding an additional section to the second level Synapse directory.
  - synapse/userguide.rst - This controls the TOC ordering for user guides.
  - synapse/devops.rst - The controls the TOC ordering for devops guides.
- Add notebooks to the repository using `git add ..path/to/notebook.ipynb`. You can then commit the notebook using `git commit`. If you have the git pre-commit hook from `scripts/githooks/pre-commit`, this will strip any output from the notebook upon commit time. This will result in cleaner `git diff` views over time. See *Git Hook & Syntax Checking*

9.2.3 Under the hood

Docs are built from Notebooks using a custom `conf.py` file which executes the notebooks, converting them to RST and using a custom template which looks for flags set by the `hide_code` extension in order to hide the blocks as needed.

9.3 Synapse Release Process

This doc details the release process we use for Synapse.

9.3.1 Github Milestone Management

The current milestone and the next milestone should be created in github. For example, if the current release is v0.2.1, we should have a v0.2.2 and v0.2.3 milestones created. When PRs are created or issues are addressed (via PR), they should be added to the milestone. This allows us to easily pull stories and PRs for release note generation.

9.3.2 Release Notes Format

Release notes should be compiled from the issues and PRs assigned to the milestone being released. These can all be obtained via a issue search in github. For example, if we're releasing v0.2.2, we can pull all the stories via the following query in github:

```
milestone:v0.2.2
```

Release notes should break things out by the following categories:

1. New Features in Synapse & Enhancements to existing features
2. Bugfixes
3. Major documentation updates

Short text form is fine for describing these.

9.3.3 Cutting the Release

This includes three parts:

1. Preparing the release notes/changelog information.
2. Tagging the release and pushing to github.
3. Close out the milestone in Github.

Preparing The Release Notes

Changelog notes are kept in the `CHANGELOG.rst` file. This allows us to keep a copy of the release notes in the repository, as well as having them automatically built into our documentation. This file needs to be updated prior to the release tagging. The formatting for adding the content to the file is the following:

```
<git tag> - YYYY-MM-DD
======================

Features and Enhancements
-------------------------

- Add new features (`#XXX <https://github.com/vertexproject/synapse/pull/XXX>`_)

Bugfixes
```

(continues on next page)

(continued from previous page)

```
--------

- Fix old bugs (`#XXX <https://github.com/vertexproject/synapse/pull/XXX>`_)

Improved Documentation
----------------------

- Write awesome docs (`#XXX <https://github.com/vertexproject/synapse/pull/XXX>`_)
```

This also allows for machine parseable notes so that `pyup.io` can show our changelogs.

It is recommended that as new PRs are made, the PR includes an update to the `CHANGELOG.rst` file so that during a release, notes don't have to be updated. If that has been done; a simple double check of the issues in the Github milestone should show anything missing.

When prepping the release, it is okay to add a blank template with the tag set to the next patch value and TBD date, so that PRs have a place to put their changelogs as they come in.

Tagging the Release

Version tagging in Synapse is managed by bumpversion. This handles updating the .py files containing the version number in them, as well as creating git tags and commit messages. There should not be a need to manually edit version numbers or do git commits.

bumpversion is a python application, and can be installed via pip:

```
python -m pip install bumpversion
```

> **Warning:** Do *not* use `bump2version`, the API compatible fork of bumpversion. It changed how tags are made which are incompatible with our current CircleCI based workflows.

Bumpversion is designed for projects which do semantic versioning. This can be done via the following (assuming the vertexproject/synapse remote is called 'upstream'):

```
# Ensure we're on master with the latest version
git checkout master && git fetch --all && git merge upstream/master
# Do a dry-run to ensure that we're updating things properly
bumpversion --dry-run --verbose patch
# Bump the patch version
bumpversion --verbose patch
# Ensure that no erroneous changes were introduced by bumvpersion
git show HEAD
# Push the new commit and tag up to github
git push upstream
# Push the new tag up explicitly. Do not use --tags
git push upstream <the new tag>
```

Next, go to github at https://github.com/vertexproject/synapse/tags and edit the release notes for the tag that was pushed up. Add a link to the release notes from the readthedocs changelog page for the current release.

Closing Milestone in Github

Close out the milestone associated with the just released version at the milestones page so no new issues are added to it.

Publishing on Pypi

Publishing packages to PyPI is done via CircleCi configuration.

Updating Docker images

Publishing docker images to DockerHub is done via CircleCi configuration.

CHAPTER

TEN

SYNAPSE PYTHON API

10.1 synapse package

The synapse intelligence analysis framework.

10.1.1 Subpackages

synapse.cmds package

Submodules

synapse.cmds.boss module

class synapse.cmds.boss.**KillCmd**(*cli*, ***opts*)

Bases: *Cmd*

Kill a running task/query within the cortex.

Syntax:
kill <iden>

Users may specify a partial iden GUID in order to kill exactly one matching process based on the partial guid.

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.cmds.boss.**PsCmd**(*cli*, ***opts*)

Bases: *Cmd*

List running tasks in the cortex.

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

synapse.cmds.cortex module

class synapse.cmds.cortex.**Log**(*cli*, ***opts*)

Bases: *Cmd*

Add a storm log to the local command session.

Notes

By default, the log file contains all messages received from the execution of a Storm query by the current CLI. By default, these messages are saved to a file located in ~/.syn/stormlogs/storm_(date).(format).

Examples

Enable logging all messages to mpk files (default) log –on

Disable logging and close the current file log –off

Enable logging, but only log edits. Log them as jsonl instead of mpk. log –on –edits-only –format jsonl

Enable logging, but log to a custom path: log –on –path /my/aweome/log/directory/storm20010203.mpk

Log only the node messages which come back from a storm cmd execution. log –on –nodes-only –path /my/awesome/log/directory/stormnodes20010203.mpk

closeLogFd()

encodeMsg(*mesg*)

Get byts for a message

onStormMesg(*mesg*)

openLogFd(*opts*)

queueLoop()

async **runCmdOpts**(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters

opts (*dict*) – Options dictionary.

save(*mesg*)

class synapse.cmds.cortex.**StormCmd**(*cli*, ***opts*)

Bases: *Cmd*

Execute a storm query.

Syntax:

storm <query>

Parameters

query – The storm query

Optional Arguments:

–hide-tags: Do not print tags. –hide-props: Do not print secondary properties. –hide-unknown: Do not print messages which do not have known handlers. –show-nodeedits: Show full nodeedits (otherwise printed as a single . per edit). –editformat <format>: What format of edits the server shall emit.

Options are

- nodeedits (default),
- count (just counts of nodeedits), or
- none (no such messages emitted).

–show-prov: Deprecated. This no longer does anything. –raw: Print the nodes in their raw format. This overrides –hide-tags and –hide-props. –debug: Display cmd debug information along with nodes in raw format. This overrides other display arguments. –path: Get path information about returned nodes. –show <names>: Limit storm events (server-side) to the comma-separated list. –file <path>: Run the storm query specified in the given file path. –optsfile <path>: Run the query with the given options from a JSON/YAML file.

Examples

storm inet:ipv4=1.2.3.4 storm –debug inet:ipv4=1.2.3.4

`editformat_enums = ('nodeedits', 'count', 'none')`

`printf`(*mesg*, *addnl=True*, *color=None*)

`async runCmdOpts`(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
`opts` (*dict*) – Options dictionary.

synapse.cmds.hive module

`class synapse.cmds.hive.HiveCmd`(*cli*, ***opts*)

Bases: *Cmd*

Manipulates values in a cell's Hive.

A Hive is a hierarchy persistent storage mechanism typically used for configuration data.

`static parsepath`(*path*)

Turn a slash-delimited path into a list that hive takes

`async runCmdOpts`(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
`opts` (*dict*) – Options dictionary.

synapse.data package

`synapse.data.get`(*name*, *defval=None*)

Return an object from the embedded synapse data folder.

Example

for tld in synapse.data.get('iana.tlds'):
    dostuff(tld)

NOTE: Files are named synapse/data/<name>.mpk

synapse.data.**getJSON**(*name*)

synapse.data.**path**(**names*)

synapse.lib package

Subpackages

synapse.lib.crypto package

Submodules

synapse.lib.crypto.coin module

synapse.lib.crypto.coin.**bch_check**(*match: Match*)

synapse.lib.crypto.coin.**btc_base58_check**(*match: Match*)

synapse.lib.crypto.coin.**btc_bech32_check**(*match: Match*)

synapse.lib.crypto.coin.**cardano_byron_check**(*match: Match*)

synapse.lib.crypto.coin.**cardano_shelly_check**(*match: Match*)

synapse.lib.crypto.coin.**eth_check**(*match: Match*)

synapse.lib.crypto.coin.**ether_eip55**(*body: str*)

synapse.lib.crypto.coin.**logger = <Logger synapse.lib.crypto.coin (WARNING)>**

synapse.lib.crypto.coin contains functions for verifying whether or not a given regex match containing a valu is valid for a given type of coin.

these functions are intended to be used with synapse.lib.scrape.

synapse.lib.crypto.coin.**substrate_check**(*match: Match*)

synapse.lib.crypto.coin.**xrp_check**(*match: Match*)

synapse.lib.crypto.ecc module

class synapse.lib.crypto.ecc.**PriKey**(*priv*)

Bases: `object`

A helper class for using ECC private keys.

dump()

Get the private key bytes in DER/PKCS8 format.

Returns
The DER/PKCS8 encoded private key.

Return type
bytes

exchange(*pubkey*)

Perform a ECDH key exchange with a public key.

Parameters
pubkey (PubKey) – A PubKey to perform the ECDH with.

Returns
The ECDH bytes. This is deterministic for a given pubkey and private key.

Return type
bytes

static **generate**()

Generate a new ECC PriKey instance.

Returns
A new PriKey instance.

Return type
PriKey

iden()

Return a SHA256 hash for the public key (to be used as a GUID).

Returns
The SHA256 hash of the public key bytes.

Return type
str

static **load**(*byts*)

Create a PriKey instance from DER/PKCS8 encoded bytes.

Parameters
byts (*bytes*) – Bytes to load

Returns
A new PubKey instance.

Return type
PriKey

public()

Get the PubKey which corresponds to the ECC PriKey.

Returns
A new PubKey object whose key corresponds to the private key.

Return type
PubKey

sign(*byts*)

Compute the ECC signature for the given bytestream.

Parameters
byts (*bytes*) – The bytes to sign.

Returns
The RSA Signature bytes.

Return type
bytes

class synapse.lib.crypto.ecc.**PubKey**(*publ*)

Bases: `object`

A helper class for using ECC public keys.

dump()

Get the public key bytes in DER/SubjectPublicKeyInfo format.

Returns
The DER/SubjectPublicKeyInfo encoded public key.

Return type
bytes

iden()

Return a SHA256 hash for the public key (to be used as a GUID).

Returns
The SHA256 hash of the public key bytes.

Return type
str

static load(*byts*)

Create a PubKey instance from DER/PKCS8 encoded bytes.

Parameters
byts (*bytes*) – Bytes to load

Returns
A new PubKey instance.

Return type
PubKey

verify(*byts*, *sign*)

Verify the signature for the given bytes using the ECC public key.

Parameters

- **byts** (*bytes*) – The data bytes.
- **sign** (*bytes*) – The signature bytes.

Returns
True if the data was verified, False otherwise.

Return type
bool

synapse.lib.crypto.ecc.**doECDHE**(*statprv_u*, *statpub_v*, *ephmprv_u*, *ephmpub_v*, *length=64*, *salt=None*, *info=None*)

Perform one side of an Ecliptic Curve Diffie Hellman Ephemeral key exchange.

Parameters

- **statprv_u** (PriKey) – Static Private Key for U
- **(PubKey** (*statpub_v*) – Static Public Key for V
- **ephmprv_u** (PriKey) – Ephemeral Private Key for U
- **ephmpub_v** (PubKey) – Ephemeral Public Key for V
- **length** (*int*) – Number of bytes to return
- **salt** (*bytes*) – Salt to use when computing the key.
- **info** (*bytes*) – Additional information to use when computing the key.

Notes

This makes no assumption about the reuse of the Ephemeral keys passed to the function. It is the caller's responsibility to destroy the keys after they are used for doing key generation. This implementation is the dhHybrid1 scheme described in NIST 800-56A Revision 2.

Returns
The derived key.

Return type
bytes

synapse.lib.crypto.passwd module

async synapse.lib.crypto.passwd.**checkShadowV2**(*passwd: AnyStr*, *shadow: Dict*) → bool

Check a password against a shadow dictionary.

Parameters

- **passwd** (*str*) – Password to check.
- **shadow** (*dict*) – Data to check the password against.

Returns
True if the password is valid, false otherwise.

Return type
bool

async synapse.lib.crypto.passwd.**generateApiKey**(*iden=None*)

async synapse.lib.crypto.passwd.**getPbkdf2**(*passwd: AnyStr*) → Dict

async synapse.lib.crypto.passwd.**getShadowV2**(*passwd: AnyStr*) → Dict

Get the shadow dictionary for a given password.

Parameters

- **passwd** (*str*) – Password to hash.
- **ptyp** (*str*) – The password hash type.

Returns
A dictionary containing shadowed password information.

Return type
dict

`synapse.lib.crypto.passwd.`**`parseApiKey`**`(`*valu*`)`

async `synapse.lib.crypto.passwd.`**`verifyPbkdf2`**`(`*passwd: AnyStr*, *shadow: Dict*`)` → bool

synapse.lib.crypto.rsa module

class `synapse.lib.crypto.rsa.`**`PriKey`**`(`*priv*`)`

Bases: `object`

A helper class for using RSA private keys.

Signing methods use RSA-PSS and MFG1 with sha256 hashing.

`iden``()` → str

Return a SHA256 hash for the public key (to be used as a GUID).

Returns
The SHA256 hash of the public key bytes.

Return type
str

`public``()`

Get the PubKey which corresponds to the RSA PriKey.

Returns
A new PubKey object whose key corresponds to the private key.

Return type
PubKey

`sign``(`*byts: bytes*`)` → bytes

Compute the RSA signature for the given bytestream.

Parameters
byts (*bytes*) – The bytes to sign.

Returns
The RSA Signature bytes.

Return type
bytes

`signitem``(`*item*`)` → bytes

Compute the RSA signature for the given python primitive.

Parameters
item – The item to sign. This will be flattened and msgpacked prior to signing.

Returns
The RSA Signature bytes.

Return type
bytes

class synapse.lib.crypto.rsa.**PubKey**(*publ*)

Bases: `object`

A helper class for using RSA public keys.

dump()

Get the public key bytes in DER/SubjectPublicKeyInfo format.

Returns
The DER/SubjectPublicKeyInfo encoded public key.

Return type
bytes

iden()

Return a SHA256 hash for the public key (to be used as a GUID).

Returns
The SHA256 hash of the public key bytes.

Return type
str

static load(*byts*)

Create a PubKey instance from DER/PKCS8 encoded bytes.

Parameters
byts (*bytes*) – Bytes to load

Returns
A new PubKey instance.

Return type
PubKey

verify(*byts*, *sign*)

Verify the signature for the given bytes using the RSA public key.

Parameters

- **byts** (*bytes*) – The data bytes.
- **sign** (*bytes*) – The signature bytes.

Returns
True if the data was verified, False otherwise.

Return type
bool

verifyitem(*item*, *sign*)

Verify the signature for the given item with the RSA public key.

Parameters

- **item** – The Python primitive to verify.
- **sign** (*bytes*) – The signature bytes.

Returns
True if the data was verified, False otherwise.

Return type
bool

synapse.lib.crypto.tinfoil module

class `synapse.lib.crypto.tinfoil.`**`CryptSeq`**(*rx_key*, *tx_key*, *initial_rx_seq=0*, *initial_tx_seq=0*)

Bases: `object`

Applies and verifies sequence numbers of encrypted messages coming and going

Parameters

- **`rx_key`** (*`bytes`*) – TX key (used with TinFoilHat).
- **`tx_key`** (*`bytes`*) – RX key (used with TinFoilHat).
- **`initial_rx_seq`** (*`int`*) – Starting rx sequence number.
- **`initial_tx_seq`** (*`int`*) – Starting tx sequence number.

`decrypt`(*ciphertext*)

Decrypt a message, validating its sequence number is as we expect.

Parameters
`ciphertext` (*`bytes`*) – The message to decrypt and verify.

Returns
A mesg.

Return type
mesg

Raises
`s_exc.CryptoErr` – If the message decryption fails or the sequence number was unexpected.

`encrypt`(*mesg*)

Wrap a message with a sequence number and encrypt it.

Parameters
`mesg` – The mesg to encrypt.

Returns
The encrypted message.

Return type
bytes

class `synapse.lib.crypto.tinfoil.`**`TinFoilHat`**(*ekey*)

Bases: `object`

The TinFoilHat class implements a GCM-AES encryption/decryption class.

Parameters

- **`ekey`** (*`bytes`*) – A 32 byte key used for doing encryption & decryption. It
- **`manner.`** (*`is assumed the caller has generated the key in a safe`*) –

`dec`(*byts*)

Decode an envelope dict and decrypt the given bytes.

Parameters
`byts` (*`bytes`*) – Bytes to decrypt.

Returns
Decrypted message.

Return type
bytes

`enc(byts, asscd=None)`

Encrypt the given bytes and return an envelope dict in msgpack form.

Parameters

- **byts** (*bytes*) – The message to be encrypted.
- **asscd** (*bytes*) – Extra data that needs to be authenticated (but not encrypted).

Returns
The encrypted message. This is a msgpacked dictionary containing the IV, ciphertext, and associated data.

Return type
bytes

`synapse.lib.crypto.tinfoil.newkey()`

Generate a new, random 32 byte key.

Returns
32 random bytes

Return type
bytes

synapse.lib.platforms package

Home for platform specific code such as thishost info.

all platform modules *must* be importable from any platform.

(guard any platform specific code with appropriate conditionals)

Submodules

synapse.lib.platforms.common module

`synapse.lib.platforms.common.daemonize()`

For unix platforms, form a new process group using fork().

`synapse.lib.platforms.common.getLibC()`

Return a ctypes reference to libc

`synapse.lib.platforms.common.getTempDir()`

`synapse.lib.platforms.common.getVolInfo(*paths)`

Retrieve volume usage info for the given path.

`synapse.lib.platforms.common.inet_ntop(afam, byts)`

`synapse.lib.platforms.common.inet_pton(afam, text)`

`synapse.lib.platforms.common.initHostInfo()`

`synapse.lib.platforms.common.setProcName(name)`

Set the process title/name for process listing.

synapse.lib.platforms.darwin module

`synapse.lib.platforms.darwin.`**`initHostInfo`**`()`

synapse.lib.platforms.freebsd module

`synapse.lib.platforms.freebsd.`**`initHostInfo`**`()`

synapse.lib.platforms.linux module

`synapse.lib.platforms.linux.`**`getAvailableMemory`**`()`

Returns the available memory of the system

`synapse.lib.platforms.linux.`**`getCurrentLockedMemory`**`()`

Return the amount of memory this process has locked

`synapse.lib.platforms.linux.`**`getFileMappedRegion`**`(`*filename*`)`

Return a tuple of address and length of a particular file memory mapped into this process

`synapse.lib.platforms.linux.`**`getMaxLockedMemory`**`()`

Returns the maximum amount of memory this process can lock

`synapse.lib.platforms.linux.`**`getSysctls`**`()`

`synapse.lib.platforms.linux.`**`getTotalMemory`**`()`

Get the total amount of memory in the system.

Notes

This attempts to get information from cgroup data before falling back to `/proc/meminfo` data.

Returns
The number of bytes of memory available in the system.

Return type
int

`synapse.lib.platforms.linux.`**`initHostInfo`**`()`

`synapse.lib.platforms.linux.`**`maximizeMaxLockedMemory`**`()`

Remove any discretionary (i.e. soft) limits

`synapse.lib.platforms.linux.`**`mlock`**`(`*address*, *length*`)`

Lock a chunk of memory to prevent it from being swapped out, raising an OSError on error

`synapse.lib.platforms.linux.`**`mmap`**`(`*address*, *length*, *prot*, *flags*, *fd*, *offset*`)`

A simple mmap context manager that releases the GIL while mapping and unmapping. It raises an OSError on error

`synapse.lib.platforms.linux.`**`munlock`**`(`*address*, *length*`)`

Unlock a chunk of memory, raising an OSError on error

synapse.lib.platforms.windows module

synapse.lib.platforms.windows.**daemonize**()

synapse.lib.platforms.windows.**getLibC**()

Override to account for python on windows not being able to find libc sometimes…

synapse.lib.platforms.windows.**initHostInfo**()

class synapse.lib.platforms.windows.**sockaddr**

Bases: Structure

ipv4

Structure/Union member

ipv6

Structure/Union member

sa_family

Structure/Union member

synapse.lib.stormlib package

Submodules

synapse.lib.stormlib.aha module

class synapse.lib.stormlib.aha.**AhaLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with AHA.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.aha.**AhaPool**(*runt*, *poolinfo*)

Bases: *StormType*

Implements the Storm API for an AHA pool.

async stormrepr()

class synapse.lib.stormlib.aha.**AhaPoolLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with AHA service pools.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.auth module

`class synapse.lib.stormlib.auth.Gate(runt, valu, path=None)`

Bases: *Prim*

Implements the Storm API for an AuthGate.

`class synapse.lib.stormlib.auth.LibAuth(runt, name=())`

Bases: *Lib*

A Storm Library for interacting with Auth in the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async getPermDef(perm)`

`async getPermDefs()`

`static ruleFromText(text)`

`async textFromRule(rule)`

`class synapse.lib.stormlib.auth.LibGates(runt, name=())`

Bases: *Lib*

A Storm Library for interacting with Auth Gates in the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormlib.auth.LibRoles(runt, name=())`

Bases: *Lib*

A Storm Library for interacting with Auth Roles in the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.auth.**LibUser**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with data about the current user.

addLibFuncs()

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.auth.**LibUsers**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Auth Users in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.auth.**Role**(*runt*, *valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Role.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

async value()

class synapse.lib.stormlib.auth.**User**(*runt*, *valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a User.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **stormrepr**()

async **value**()

class synapse.lib.stormlib.auth.**UserJson**(*runt*, *valu*)

Bases: *Prim*

Implements per-user JSON storage.

async **get**(*path*, *prop=None*)

async **has**(*path*)

async **iter**(*path=None*)

async **set**(*path*, *valu*, *prop=None*)

class synapse.lib.stormlib.auth.**UserProfile**(*runt*, *valu*, *path=None*)

Bases: *Prim*

The Storm deref/setitem/iter convention on top of User profile information.

async **deref**(*name*)

async **iter**()

async **setitem**(*name*, *valu*)

async **value**()

class synapse.lib.stormlib.auth.**UserVars**(*runt*, *valu*, *path=None*)

Bases: *Prim*

The Storm deref/setitem/iter convention on top of User vars information.

async **deref**(*name*)

async **iter**()

async **setitem**(*name*, *valu*)

synapse.lib.stormlib.auth.**ruleFromText**(*text*)

Get a rule tuple from a text string.

Parameters

text (*str*) – The string to process.

Returns

A tuple containing a bool and a list of permission parts.

Return type

(bool, tuple)

synapse.lib.stormlib.backup module

class synapse.lib.stormlib.backup.**BackupLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with the backup APIs in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.basex module

class synapse.lib.stormlib.basex.**BaseXLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements helpers for encoding and decoding strings using an arbitrary charset.

async decode(*text*, *charset*)

async encode(*byts*, *charset*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.cache module

class synapse.lib.stormlib.cache.**FixedCache**(*runt*, *query*, *size=10000*)

Bases: *StormType*

A StormLib API instance of a Storm Fixed Cache.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

class synapse.lib.stormlib.cache.**LibCache**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Cache Objects.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.cell module

class synapse.lib.stormlib.cell.**CellLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.cell.**getMaxHotFixes**()

synapse.lib.stormlib.compression module

class synapse.lib.stormlib.compression.**Bzip2Lib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements helpers for bzip2 compression.

async en(*valu*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async un(*valu*)

class synapse.lib.stormlib.compression.**GzipLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements helpers for gzip compression.

async en(*valu*)

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async un(`*valu*`)`

`class synapse.lib.stormlib.compression.ZlibLib(`*runt*, *name=()*`)`

Bases: *Lib*

A Storm library which implements helpers for zlib compression.

`async en(`*valu*`)`

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async un(`*valu*`)`

synapse.lib.stormlib.cortex module

`class synapse.lib.stormlib.cortex.CortexHttpApi(`*runt*, *name=()*`)`

Bases: *Lib*

Library for interacting with the Extended HTTP API.

`async addHttpApi(`*path*, *name=''*, *desc=''*, *runas='owner'*, *authenticated=True*, *readonly=False*`)`

`async delHttpApi(`*iden*`)`

`async getHttpApi(`*iden*`)`

`async getHttpApiByPath(`*path*`)`

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async listHttpApis()`

`async makeHttpResponse(`*requestinfo*`)`

`async setHttpApiIndx(`*iden*, *index=0*`)`

class synapse.lib.stormlib.cortex.**HttpApi**(*runt*, *info*)

Bases: *StormType*

Extended HTTP API object.

This object represents an extended HTTP API that has been configured on the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

class synapse.lib.stormlib.cortex.**HttpApiMethods**(*httpapi: HttpApi*)

Bases: *Prim*

Accessor dictionary for getting and setting Extended HTTP API methods.

Notes:
The Storm code used to run these methods will have a $request object injected into them. This allows the method to send data back to the caller when it is run.

Examples:
Setting a simple GET method:

```
$api.methods.get = ${
    $data = ({"someKey": "someValue})
    $headers = ({"someHeader": "someOtherValue"})
    $request.reply(200, headers=$headers, body=$data)
}
```

Removing a PUT method:

```
$api.methods.put = $lib.undef
```

Crafting a custom text response:

```
$api.methods.get = ${
    // Create the body
    $data = 'some value'
    // Encode the response as bytes
    $data = $data.encode()
    // Set the headers
    $headers = ({"Content-Type": "text/plain", "Content-Length": $lib.
↪len($data})
    $request.reply(200, headers=$headers, body=$data)
}
```

Streaming multiple chunks of data as JSON lines. This sends the code, headers and body separately:

```
$api.methods.get = ${
    $request.sendcode(200)
```

(continues on next page)

(continued from previous page)

```
    $request.sendheaders(({"Content-Type": "text/plain; charset=utf8"}))
    $values = ((1), (2), (3))
    for $i in $values {
        $body=`{$lib.json.save(({"value": $i}))}
```

`

$request.sendbody($body.encode())

}

}

async iter()

class synapse.lib.stormlib.cortex.**HttpApiVars**(*httpapi*, *path=None*)

Bases: *Dict*

Accessor dictionary for getting and setting Extended HTTP API variables.

This can be used to set, unset or iterate over the runtime variables that are set for an Extended HTTP API endpoint. These variables are set in the Storm runtime for all of the HTTP methods configured to be executed by the endpoint.

Example

Set a few variables on a given API:

```
$api.vars.foo = 'the foo string'
$api.vars.bar = (1234)
```

Remove a variable:

```
$api.vars.foo = $lib.undef
```

Iterate over the variables set for the endpoint:

```
for ($key, $valu) in $api.vars {
    $lib.print(`{$key) -> {$valu}`)
}
```

Overwrite all of the variables for a given API with a new dictionary:

```
$api.vars = ({"foo": "a new string", "bar": (137)})
```

async setitem(*name*, *valu*)

class synapse.lib.stormlib.cortex.**HttpHeaderDict**(*valu*, *path=None*)

Bases: *Dict*

Immutable lowercase key access dictionary for HTTP request headers.

Example

Request headers can be accessed in a case insensitive manner:

```
$valu = $request.headers.Cookie
// or the lower case value
$valu = $request.headers.cookie
```

async deref(*name*)

async setitem(*name*, *valu*)

class synapse.lib.stormlib.cortex.**HttpPermsList**(*httpapi*, *path=None*)

Bases: *List*

Accessor list for getting and setting http:api permissions.

async setitem(*name*, *valu*)

class synapse.lib.stormlib.cortex.**HttpReq**(*runt*, *rnfo*)

Bases: *StormType*

Extended HTTP API Request object.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.cortex.**StormPoolDelCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Remove a Storm query offload mirror pool configuration.

Notes

This will result in tearing down any Storm queries currently being serviced by the Storm pool. This may result in this command raising an exception if it was offloaded to a pool member. That would be an expected behavior.

async execStormCmd(*runt*, *genr*)

Abstract base method

name = 'cortex.storm.pool.del'

class synapse.lib.stormlib.cortex.**StormPoolGetCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Display the current Storm query offload mirror pool configuration.

async execStormCmd(*runt*, *genr*)

Abstract base method

name = 'cortex.storm.pool.get'

class synapse.lib.stormlib.cortex.**StormPoolSetCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Setup a Storm query offload mirror pool for the Cortex.

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'cortex.storm.pool.set'

synapse.lib.stormlib.easyperm module

class synapse.lib.stormlib.easyperm.**LibEasyPerm**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with easy perm dictionaries.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.ethereum module

class synapse.lib.stormlib.ethereum.**EthereumLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements helpers for Ethereum.

async eip55(*addr*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.gen module

class synapse.lib.stormlib.gen.**LibGen**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for secondary property based deconfliction.

synapse.lib.stormlib.gis module

class `synapse.lib.stormlib.gis.GisLib(runt, name=())`

Bases: *Lib*

A Storm library which implements helpers for earth based geospatial calculations.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.graph module

class `synapse.lib.stormlib.graph.GraphLib(runt, name=())`

Bases: *Lib*

A Storm Library for interacting with graph projections in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.hashes module

class `synapse.lib.stormlib.hashes.LibHashes(runt, name=())`

Bases: *Lib*

A Storm Library for hashing bytes

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class `synapse.lib.stormlib.hashes.LibHmac(runt, name=())`

Bases: *Lib*

A Storm library for computing RFC2104 HMAC values.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.hex module

class synapse.lib.stormlib.hex.**HexLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements helpers for hexadecimal encoded strings.

async **decode**(*valu*)

async **encode**(*valu*)

async **fromint**(*valu*, *length*, *signed=False*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **signext**(*valu*, *length*)

async **toint**(*valu*, *signed=False*)

async **trimext**(*valu*)

synapse.lib.stormlib.imap module

class synapse.lib.stormlib.imap.**ImapLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library to connect to an IMAP server.

async **connect**(*host*, *port=993*, *timeout=30*, *ssl=True*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.imap.**ImapServer**(*runt*, *imap_cli*, *path=None*)

Bases: *StormType*

An IMAP server for retrieving email messages.

async **delete**(*uid_set*)

async **fetch**(*uid*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **list**(*reference_name='""', pattern='*'*)

async **login**(*user, passwd*)

async **markSeen**(*uid_set*)

async **search**(**args, charset='utf-8'*)

async **select**(*mailbox='INBOX'*)

async synapse.lib.stormlib.imap.**run_imap_coro**(*coro*)

Raises or returns data.

synapse.lib.stormlib.infosec module

synapse.lib.stormlib.infosec.**CVSS2_calc**(*vdict*)

synapse.lib.stormlib.infosec.**CVSS2_round**(*x*)

synapse.lib.stormlib.infosec.**CVSS3_0_calc**(*vdict*)

synapse.lib.stormlib.infosec.**CVSS3_0_round**(*x*)

Round up to the nearest one decimal place. From the JS reference implementation: https://www.first.org/cvss/calculator/cvsscalc30.js

synapse.lib.stormlib.infosec.**CVSS3_1_calc**(*vdict*)

synapse.lib.stormlib.infosec.**CVSS3_1_round**(*x*)

Round up to the nearest one decimal place. From the JS reference implementation: https://www.first.org/cvss/calculator/cvsscalc31.js

synapse.lib.stormlib.infosec.**CVSS_get_coefficients**(*vdict, vers*)

class synapse.lib.stormlib.infosec.**CvssLib**(*runt, name=()*)

Bases: `Lib`

A Storm library which implements CVSS score calculations.

async **calculate**(*node, save=True, vers='3.1'*)

async **calculateFromProps**(*props, vers='3.1'*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **saveVectToNode**(*node*, *text*)

async **vectToProps**(*text*)

async **vectToScore**(*vect*, *vers=None*)

class synapse.lib.stormlib.infosec.**MitreAttackFlowLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements modeling MITRE ATT&CK Flow diagrams.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.infosec.**roundup**(*x*)

synapse.lib.stormlib.ipv6 module

class synapse.lib.stormlib.ipv6.**LibIpv6**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for providing ipv6 helpers.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.iters module

class synapse.lib.stormlib.iters.**LibIters**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for providing iterator helpers.

async **enum**(*genr*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.json module

class synapse.lib.stormlib.json.**JsonLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Json data.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.json.**JsonSchema**(*runt*, *schema*, *use_default=True*)

Bases: *StormType*

A JsonSchema validation object for use in validating data structures in Storm.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

synapse.lib.stormlib.json.**compileJsSchema**(*schema*, *use_default=True*)

synapse.lib.stormlib.json.**runJsSchema**(*schema*, *item*, *use_default=True*)

synapse.lib.stormlib.log module

class synapse.lib.stormlib.log.**LoggerLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library which implements server side logging. These messages are logged to the `synapse.storm.log` logger.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.macro module

class synapse.lib.stormlib.macro.**LibMacro**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with the Storm Macros in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.macro.**MacroExecCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute a named macro.

Example

inet:ipv4#cno.threat.t80 | macro.exec enrich_foo

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'macro.exec'

readonly = True

synapse.lib.stormlib.math module

class synapse.lib.stormlib.math.**MathLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for performing math operations.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.mime module

class synapse.lib.stormlib.mime.**LibMimeHtml**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for manipulating HTML text.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async totext(*html*)

synapse.lib.stormlib.mime.**htmlToText**(*html*)

synapse.lib.stormlib.model module

class synapse.lib.stormlib.model.**LibModel**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with the Data Model in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.model.**LibModelDeprecated**(*runt*, *name=()*)

Bases: *Lib*

A storm library for interacting with the model deprecation mechanism.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.model.**LibModelEdge**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with light edges and manipulating their key-value attributes.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

hivepath = ('cortex', 'model', 'edges')

validedgekeys = ('doc',)

class synapse.lib.stormlib.model.**LibModelTags**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with tag specifications in the Cortex Data Model.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.model.**ModelForm**(*form*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Form.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

value()

class synapse.lib.stormlib.model.**ModelProp**(*prop*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Property.

value()

class synapse.lib.stormlib.model.**ModelTagProp**(*tagprop*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Tag Property.

value()

class synapse.lib.stormlib.model.**ModelType**(*valu*, *path=None*)

Bases: *Prim*

A Storm types wrapper around a lib.types.Type

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

value()

synapse.lib.stormlib.modelext module

class synapse.lib.stormlib.modelext.**LibModelExt**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for manipulating extended model elements.

async **addExtModel**(*model*)

async **addForm**(*formname*, *basetype*, *typeopts*, *typeinfo*)

async **addFormProp**(*formname*, *propname*, *typedef*, *propinfo*)

async **addTagProp**(*propname*, *typedef*, *propinfo*)

async **addUnivProp**(*propname*, *typedef*, *propinfo*)

async **delForm**(*formname*)

async **delFormProp**(*formname*, *propname*)

async **delTagProp**(*propname*)

async **delUnivProp**(*propname*)

async **getExtModel**()

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.notifications module

class synapse.lib.stormlib.notifications.**NotifyLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for a user interacting with their notifications.

async **get**(*indx*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async list(*size=None*)

synapse.lib.stormlib.oauth module

class synapse.lib.stormlib.oauth.**OAuthV1Client**(*runt*, *ckey*, *csecret*, *atoken*, *asecret*, *sigtype*, *path=None*)

Bases: *StormType*

A client for doing OAuth V1 Authentication from Storm.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.oauth.**OAuthV1Lib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library to handle OAuth v1 authentication.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.oauth.**OAuthV2Lib**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for managing OAuth V2 clients.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.pack module

class synapse.lib.stormlib.pack.**LibPack**(*runt*, *name=()*)

Bases: *Lib*

Packing / unpacking structured bytes.

async en(*fmt*, *items*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async un(*fmt*, *byts*, *offs=0*)

synapse.lib.stormlib.project module

class synapse.lib.stormlib.project.**LibProjects**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Projects in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async iter()

class synapse.lib.stormlib.project.**Project**(*runt*, *node*, *path=None*)

Bases: *Prim*

Implements the Storm API for Project objects, which are used for managing a scrum style project in the Cortex

confirm(*perm*)

async nodes()

value()

class synapse.lib.stormlib.project.**ProjectEpic**(*proj*, *node*)

Bases: *Prim*

Implements the Storm API for a ProjectEpic

async nodes()

async value()

class synapse.lib.stormlib.project.**ProjectEpics**(*proj*)

Bases: *Prim*

Implements the Storm API for ProjectEpics objects, which are collections of ProjectEpic objects associated with a particular Project

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async iter()`

`class synapse.lib.stormlib.project.ProjectSprint(proj, node)`

Bases: *Prim*

Implements the Storm API for a ProjectSprint

`async nodes()`

`async value()`

`class synapse.lib.stormlib.project.ProjectSprints(proj)`

Bases: *Prim*

Implements the Storm API for ProjectSprints objects, which are collections of sprints associated with a single project

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async iter()`

`class synapse.lib.stormlib.project.ProjectTicket(proj, node)`

Bases: *Prim*

Implements the Storm API for a ProjectTicket.

`async nodes()`

`async value()`

`class synapse.lib.stormlib.project.ProjectTicketComment(ticket, node)`

Bases: *Prim*

Implements the Storm API for a ProjectTicketComment

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async nodes()`

`async value()`

`class synapse.lib.stormlib.project.ProjectTicketComments(ticket)`

Bases: *Prim*

Implements the Storm API for ProjectTicketComments objects, which are collections of comments associated with a ticket.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async iter()`

`class synapse.lib.stormlib.project.ProjectTickets(proj)`

Bases: *Prim*

Implements the Storm API for ProjectTickets objects, which are collections of tickets associated with a project

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async iter()`

synapse.lib.stormlib.random module

`class synapse.lib.stormlib.random.LibRandom(runt, name=())`

Bases: *Lib*

A Storm library for generating random values.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.scrape module

`class synapse.lib.stormlib.scrape.LibScrape(runt, name=())`

Bases: *Lib*

A Storm Library for providing helpers for scraping nodes from text.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.smtp module

class synapse.lib.stormlib.smtp.**SmtpLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for sending email messages via SMTP.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async message()

class synapse.lib.stormlib.smtp.**SmtpMessage**(*runt*)

Bases: *StormType*

An SMTP message to compose and send.

async send(*host*, *port=25*, *user=None*, *passwd=None*, *usetls=False*, *starttls=False*, *timeout=60*)

synapse.lib.stormlib.spooled module

class synapse.lib.stormlib.spooled.**LibSpooled**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Spooled Objects.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.spooled.**SpooledSet**(*valu*, *path=None*)

Bases: *Set*

A StormLib API instance of a Storm Set object that can fallback to lmdb.

async iter()

async stormrepr()

async value()

synapse.lib.stormlib.stats module

class synapse.lib.stormlib.stats.**LibStats**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for statistics related functionality.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async tally()

class synapse.lib.stormlib.stats.**StatTally**(*path=None*)

Bases: *Prim*

A tally object.

An example of using it:

```
$tally = $lib.stats.tally()

$tally.inc(foo)

for $name, $total in $tally {
    $doStuff($name, $total)
}
```

async get(*name*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async inc(*name*, *valu=1*)

async iter()

async sorted(*byname=False*, *reverse=False*)

value()

class synapse.lib.stormlib.stats.**StatsCountByCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Tally occurrences of values and display a bar chart of the results.

Examples

// Show counts of geo:name values referenced by media:news nodes. media:news -(refs)> geo:name | stats.countby

// Show counts of ASN values in a set of IPs. inet:ipv4#myips | stats.countby :asn

// Show counts of attacker names for risk:compromise nodes. risk:compromise | stats.countby :attacker::name

`async execStormCmd(runt, genr)`
: Abstract base method

`getArgParser()`

`name = 'stats.countby'`

`readonly = True`

synapse.lib.stormlib.stix module

`class synapse.lib.stormlib.stix.LibStix(runt, name=())`

Bases: `Lib`

A Storm Library for interacting with Stix Version 2.1 CS02.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async liftBundle(bundle)`

`async validateBundle(bundle)`

`class synapse.lib.stormlib.stix.LibStixExport(runt, name=())`

Bases: `Lib`

A Storm Library for exporting to STIX version 2.1 CS02.

`async bundle(config=None)`

`async config()`

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`timestamp(tick)`

class synapse.lib.stormlib.stix.**LibStixImport**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for importing Stix Version 2.1 data.

async config()

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async ingest(*bundle*, *config=None*)

class synapse.lib.stormlib.stix.**StixBundle**(*libstix*, *runt*, *config*, *path=None*)

Bases: *Prim*

Implements the Storm API for creating and packing a STIX bundle for v2.1

async add(*node*, *stixtype=None*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

pack()

size()

async value()

synapse.lib.stormlib.stix.**uuid4**(*valu=None*)

synapse.lib.stormlib.stix.**uuid5**(*valu=None*)

synapse.lib.stormlib.stix.**validateStix**(*bundle*, *version='2.1'*)

synapse.lib.stormlib.storm module

class synapse.lib.stormlib.storm.**LibStorm**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for evaluating dynamic storm expressions.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.stormlib.vault module

class synapse.lib.stormlib.vault.**LibVault**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with vaults.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormlib.vault.**Vault**(*runt*, *valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Vault.

Callers (instantiation) of this class must have already checked that the user has at least PERM_READ to the vault.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

value()

class synapse.lib.stormlib.vault.**VaultConfigs**(*runt*, *valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for Vault data. This is used for both vault configs and vault secrets.

async deref(*name*)

async iter()

async setitem(*name*, *valu*)

async stormrepr()

value()

class synapse.lib.stormlib.vault.**VaultSecrets**(*runt*, *valu*, *path=None*)

Bases: *VaultConfigs*

async setitem(*name*, *valu*)

synapse.lib.stormlib.version module

class synapse.lib.stormlib.version.**VersionLib**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with version information.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async matches(*vertup*, *reqstr*)

synapse.lib.stormlib.xml module

class synapse.lib.stormlib.xml.**LibXml**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for parsing XML.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async parse(*valu*)

class synapse.lib.stormlib.xml.**XmlElement**(*runt*, *elem*)

Bases: *Prim*

A Storm object for dealing with elements in an XML tree.

async find(*name*, *nested=True*)

async get(*name*)

async iter()

synapse.lib.stormlib.yaml module

class synapse.lib.stormlib.yaml.**LibYaml**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for saving/loading YAML data.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **load**(*valu*)

async **save**(*valu*, *sort_keys=True*)

Submodules

synapse.lib.agenda module

class synapse.lib.agenda.**Agenda**

Bases: *Base*

Organize and execute all the scheduled storm queries in a cortex.

async **add**(*cdef*)

Persistently adds an appointment

Parameters
cdef (*dict*) – Dictionary containing the Cron definition.

Notes

The cron definition may contain the following keys:

creator (str)
Iden of the creating user.

iden (str)
Iden of the appointment.

storm (str)
The Storm query to run.

reqs (Union[None, Dict[TimeUnit, Union[int, Tuple[int]], List[...])
One or more dicts of the fixed aspects of the appointment. dict value may be a single or multiple. May be an empty dict or None.

incunit (Union[None, TimeUnit])
The unit that changes for recurring, or None for non-recurring. It is an error for this value to match a key in reqdict.

incvals (Union[None, int, Iterable[int])
Count of units of incunit or explicit day of week or day of month. Not allowed for incunit == None, required for others (1 would be a typical value)

If the values for req and incvals are both lists, all combinations of all values (the product) are used.

Returns
Packed appointment definition

async clearRunningStatus()

Used for clearing the running state at startup or change of leadership.

async delete(*iden*)

Delete an appointment

async disable(*iden*)

async enable(*iden*)

async get(*iden*)

list()

async mod(*iden*, *query*)

Change the query of an appointment

async move(*croniden*, *viewiden*)

Move a cronjob from one view to another

async runloop()

Task loop to issue query tasks at the right times.

class synapse.lib.agenda.ApptRec(*reqdict*, *incunit=None*, *incval=1*)

Bases: `object`

Represents a single element of a single combination of an appointment

nexttime(*lastts*)

Returns next timestamp that meets requirements, incrementing by (self.incunit * incval) if not increasing, or 0.0 if there are no future matches

pack()

Make ApptRec json/msgpack-friendly

classmethod unpack(*val*)

Convert from json/msgpack-friendly

class synapse.lib.agenda.TimeUnit(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntEnum`

Unit of time that recurring and required parts of appointments are made of

DAY = 5

DAYOFMONTH = 3

DAYOFWEEK = 4

HOUR = 6

MINUTE = 7

MONTH = 2

NOW = 8

YEAR = 1

classmethod fromString(*s*)

synapse.lib.aha module

class synapse.lib.aha.**AhaApi**

Bases: *CellApi*

addAhaPool(*name*, *info*)

addAhaPoolSvc(*poolname*, *svcname*, *info*)

async addAhaSvc(*name*, *info*, *network=None*)

Register a service with the AHA discovery server.

NOTE: In order for the service to remain marked "up" a caller
must maintain the telepath link.

addAhaSvcProv(*name*, *provinfo=None*)

Provision the given relative service name within the configured network name.

addAhaUserEnroll(*name*, *userinfo=None*, *again=False*)

Create and return a one-time user enroll key.

clearAhaSvcProvs()

Remove all unused service provisioning values.

clearAhaUserEnrolls()

Remove all unused user enrollment provisioning values.

delAhaPool(*name*)

delAhaPoolSvc(*poolname*, *svcname*)

async delAhaSvc(*name*, *network=None*)

Remove an AHA service entry.

delAhaSvcProv(*iden*)

Remove a previously added provisioning entry by iden.

delAhaUserEnroll(*iden*)

Remove a previously added enrollment entry by iden.

async genCaCert(*network*)

async getAhaPool(*name*)

async getAhaPools()

async getAhaSvc(*name*, *filters=None*)

Return an AHA service description dictionary for a service name.

async getAhaSvcMirrors(*name*)

Return list of AHA svcinfo dictionaries for mirrors of a service.

async getAhaSvcs(*network=None*)

Yield AHA svcinfo dictionaries.

Parameters

network (*str*) – Optionally specify a network to filter on.

async getAhaUrls()

async **getCaCert**(*network*)

async **iterPoolTopo**(*name*)

async **modAhaSvcInfo**(*name*, *svcinfo*)

async **signHostCsr**(*csrtext*, *signas=None*, *sans=None*)

async **signUserCsr**(*csrtext*, *signas=None*)

class synapse.lib.aha.**AhaCell**

Bases: *Cell*

async **addAhaPool**(*name*, *info*)

async **addAhaPoolSvc**(*poolname*, *svcname*, *info*)

async **addAhaSvc**(*name*, *info*, *network=None*)

async **addAhaSvcProv**(*name*, *provinfo=None*)

async **addAhaUserEnroll**(*name*, *userinfo=None*, *again=False*)

cellapi

alias of *AhaApi*

async **clearAhaSvcProvs**()

async **clearAhaUserEnrolls**()

confbase = {'_log_conf': {'description': 'Opaque structure used for logging by spawned processes.', 'hideconf': True, 'type': 'object'}, 'aha:admin': {'description': 'An AHA client certificate CN to register as a local admin user.', 'type': 'string'}, 'aha:leader': {'description': 'The AHA service name to claim as the active instance of a storm service.', 'type': 'string'}, 'aha:name': {'description': 'The name of the cell service in the aha service registry.', 'type': 'string'}, 'aha:network': {'description': 'The AHA service network. This makes aha:name/aha:leader relative names.', 'type': 'string'}, 'aha:provision': {'description': 'The telepath URL of the aha provisioning service.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:registry': {'description': 'The telepath URL of the aha service registry.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:svcinfo': {'description': 'An AHA svcinfo object. If set, this overrides self discovered Aha service information.', 'hidecmdl': True, 'hidedocs': True, 'properties': {'urlinfo': {'properties': {'host': {'type': 'string'}, 'port': {'type': 'integer'}, 'schema': {'type': 'string'}}, 'required': ('host', 'port', 'scheme'), 'type': 'object'}}, 'required': ('urlinfo',), 'type': 'object'}, 'aha:user': {'description': 'The username of this service when connecting to others.', 'type': 'string'}, 'auth:anon': {'description': 'Allow anonymous telepath access by mapping to the given user name.', 'type': 'string'}, 'auth:conf': {'description': 'Extended configuration to be used by an alternate auth constructor.', 'hideconf': True, 'type': 'object'}, 'auth:ctor': {'description': 'Allow the construction of the cell auth object to be hooked at runtime.', 'hideconf': True, 'type': 'string'}, 'auth:passwd': {'description': 'Set to <passwd> (local only) to bootstrap the root user password.', 'type': 'string'}, 'backup:dir': {'description': 'A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.', 'type': 'string'}, 'cell:ctor': {'description': 'An optional python path to the Cell class. Used by stemcell.', 'hideconf': True, 'type': 'string'}, 'cell:guid': {'description': 'An optional hard-coded GUID to store as the permanent GUID for the service.', 'hideconf': True, 'type': 'string'}, 'dmon:listen': {'description': 'A config-driven way to specify the telepath bind URL.', 'type': ['string', 'null']}, 'https:headers': {'description': 'Headers to add to all HTTPS server responses.', 'hidecmdl': True, 'type': 'object'}, 'https:parse:proxy:remoteip': {'default': False, 'description': 'Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.', 'type': 'boolean'}, 'https:port': {'description': 'A config-driven way to specify the HTTPS port.', 'type': ['integer', 'null']}, 'inaugural': {'description': 'Data used to drive configuration of the service upon first startup.', 'hidedocs': True, 'properties': {'roles': {'items': {'additionalProperties': False, 'properties': {'name': {'pattern': '^(?!all$).+$', 'type': 'string'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}, 'users': {'items': {'additionalProperties': False, 'properties': {'admin': {'default': False, 'type': 'boolean'}, 'email': {'type': 'string'}, 'name': {'pattern': '^(?!root$).+$', 'type': 'string'}, 'roles': {'items': {'type': 'string'}, 'type': 'array'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}}, 'type': 'object'}, 'limit:disk:free': {'default': 5, 'description': 'Minimum disk free space percentage before setting the cell read-only.', 'maximum': 100, 'minimum': 0, 'type': ['integer', 'null']}, 'max:users': {'default': 0, 'description': 'Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).', 'minimum': 0, 'type': 'integer'}, 'mirror': {'description': 'A telepath URL for our upstream mirror (we must be a backup!).', 'hidecmdl': False, 'hidedocs': False, 'type': ['string', 'null']}, 'nexslog:async': {'default': True, 'description': 'Set to false to disable async memory mapping of the nexus change log.', 'hidecmdl': True, 'hidedocs': True, 'type': 'boolean'}, 'nexslog:en': {'default': False, 'description': 'Record all

confdefs = {'aha:urls': {'description': 'A list of all available AHA server URLs.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'provision:listen': {'description': 'A telepath URL for the AHA provisioning listener.', 'type': ['string', 'null']}}

async **delAhaPool**(*name*)

async **delAhaPoolSvc**(*poolname*, *svcname*)

async **delAhaSvc**(*name*, *network=None*)

async **delAhaSvcProv**(*iden*)

async **delAhaUserEnroll**(*iden*)

async **genCaCert**(*network*)

async **getAhaPool**(*name*)

async **getAhaPools**()

async **getAhaSvc**(*name*, *filters=None*)

async **getAhaSvcMirrors**(*iden*, *network=None*)

async **getAhaSvcProv**(*iden*)

async **getAhaSvcs**(*network=None*)

async **getAhaUserEnroll**(*iden*)

async **getCaCert**(*network*)

classmethod **getEnvPrefix**()

Get a list of envar prefixes for config resolution.

async **initServiceNetwork**()

async **initServiceRuntime**()

async **initServiceStorage**()

async **iterPoolTopo**(*name*)

async **modAhaSvcInfo**(*name*, *svcinfo*)

async **saveCaCert**(*name*, *cakey*, *cacert*)

async **saveHostCert**(*name*, *hostkey*, *hostcert*)

async **saveUserCert**(*name*, *userkey*, *usercert*)

async **setAhaSvcDown**(*name*, *linkiden*, *network=None*)

async **signHostCsr**(*csrtext*, *signas=None*, *sans=None*)

async **signUserCsr**(*csrtext*, *signas=None*)

class synapse.lib.aha.**AhaProvisionServiceV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.aha.**AhaServicesV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async get()

class synapse.lib.aha.**EnrollApi**(*aha*, *userinfo*)

Bases: object

async getCaCert()

async getUserInfo()

async signUserCsr(*byts*)

class synapse.lib.aha.**ProvApi**(*aha*, *provinfo*)

Bases: object

async getCaCert()

async getProvInfo()

async signHostCsr(*byts*)

async signUserCsr(*byts*)

class synapse.lib.aha.**ProvDmon**

Bases: *Daemon*

synapse.lib.ast module

class synapse.lib.ast.**AbsPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**AndCond**(*astinfo*, *kids=()*)

Bases: *Cond*

<cond> and <cond>

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

async getLiftHints(*runt*, *path*)

class synapse.lib.ast.**ArgvQuery**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

isRuntSafe(*runt*)

runtopaque = True

validate(*runt*)

class synapse.lib.ast.ArrayCond(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.AstNode(*astinfo*, *kids=()*)

Bases: object

Base class for all nodes in the Storm abstract syntax tree.

addExcInfo(*exc*)

addKid(*astn*)

format(*depth=0*)

getAstText()

getPosInfo()

getRuntVars(*runt*)

hasAstClass(*clss*)

hasVarName(*name*)

init(*core*)

isRuntSafe(*runt*)

isRuntSafeAtom(*runt*)

iterright()

Yield "rightward" siblings until None.

optimize()

prepare()

repr()

reqRuntSafe(*runt*, *mesg*)

runtopaque = False

sibling(*offs=1*)

Return sibling node by relative offset from self.

validate(*runt*)

class synapse.lib.ast.**Bool**(*astinfo*, *valu*, *kids=()*)
Bases: `Const`

class synapse.lib.ast.**BreakOper**(*astinfo*, *kids=()*)
Bases: `AstNode`

async run(*runt*, *genr*)

class synapse.lib.ast.**CallArgs**(*astinfo*, *kids=()*)
Bases: `Value`

async compute(*runt*, *path*)

class synapse.lib.ast.**CallKwarg**(*astinfo*, *kids=()*)
Bases: `CallArgs`

class synapse.lib.ast.**CallKwargs**(*astinfo*, *kids=()*)
Bases: `CallArgs`

class synapse.lib.ast.**CaseEntry**(*astinfo*, *kids=()*)
Bases: `AstNode`

class synapse.lib.ast.**CatchBlock**(*astinfo*, *kids=()*)
Bases: `AstNode`

async catches(*name*, *runt*, *path=None*)

errvar()

getRuntVars(*runt*)

async run(*runt*, *genr*)

class synapse.lib.ast.**CmdOper**(*astinfo*, *kids=()*)
Bases: `Oper`

async run(*runt*, *genr*)

class synapse.lib.ast.**Cmpr**(*astinfo*, *valu*, *kids=()*)
Bases: `Const`

class synapse.lib.ast.**Cond**(*astinfo*, *kids=()*)
Bases: `Value`

A condition that is evaluated to filter nodes.

class synapse.lib.ast.**Const**(*astinfo*, *valu*, *kids=()*)
Bases: `Value`

async compute(*runt*, *path*)

isRuntSafe(*runt*)

repr()

value()

class synapse.lib.ast.**ContinueOper**(*astinfo*, *kids=()*)
Bases: `AstNode`

async run(*runt*, *genr*)

class synapse.lib.ast.**DollarExpr**(*astinfo*, *kids=()*)

Bases: *Value*

Top level node for $(…) expressions

async compute(*runt*, *path*)

class synapse.lib.ast.**Edit**(*astinfo*, *kids=()*)

Bases: *Oper*

class synapse.lib.ast.**EditEdgeAdd**(*astinfo*, *kids=()*, *n2=False*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditEdgeDel**(*astinfo*, *kids=()*, *n2=False*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditNodeAdd**(*astinfo*, *kids=()*)

Bases: *Edit*

async addFromPath(*form*, *runt*, *path*)

Add a node using the context from path.

NOTE: CALLER MUST CHECK PERMS

prepare()

async run(*runt*, *genr*)

class synapse.lib.ast.**EditParens**(*astinfo*, *kids=()*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditPropDel**(*astinfo*, *kids=()*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditPropSet**(*astinfo*, *kids=()*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditTagAdd**(*astinfo*, *kids=()*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditTagDel**(*astinfo*, *kids=()*)

Bases: *Edit*

async run(*runt*, *genr*)

class synapse.lib.ast.**EditTagPropDel**(*astinfo*, *kids=()*)

Bases: *Edit*

[-#foo.bar:baz]

async **run**(*runt*, *genr*)

class synapse.lib.ast.**EditTagPropSet**(*astinfo*, *kids=()*)

Bases: *Edit*

[#foo.bar:baz=10]

async **run**(*runt*, *genr*)

class synapse.lib.ast.**EditUnivDel**(*astinfo*, *kids=()*)

Bases: *Edit*

async **run**(*runt*, *genr*)

class synapse.lib.ast.**EmbedQuery**(*astinfo*, *valu*, *kids=()*)

Bases: *Const*

async **compute**(*runt*, *path*)

getRuntVars(*runt*)

hasVarName(*name*)

runtopaque = True

validate(*runt*)

class synapse.lib.ast.**Emit**(*astinfo*, *kids=()*)

Bases: *Oper*

async **run**(*runt*, *genr*)

class synapse.lib.ast.**EmptyBlock**(*astinfo*, *kids=()*)

Bases: *AstNode*

An AST node that only runs if there are not inbound nodes in the pipeline. It is capable of yielding nodes into the pipeline.

Example

Using an empty block:

```
empty {
    // the pipeline is empty so this block will execute
}

[foo:bar=*]
empty {
    // there is a node in the pipeline so this block will not run
}
```

async **run**(*runt*, *genr*)

class synapse.lib.ast.**ExprAndNode**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**ExprDict**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

prepare()

class synapse.lib.ast.**ExprList**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

prepare()

class synapse.lib.ast.**ExprNode**(*astinfo*, *kids=()*)

Bases: *Value*

A binary (i.e. two argument) expression node

async **compute**(*runt*, *path*)

prepare()

class synapse.lib.ast.**ExprOrNode**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**FiltByArray**(*astinfo*, *kids=()*)

Bases: *FiltOper*

+:foo*[^=visi]

class synapse.lib.ast.**FiltOper**(*astinfo*, *kids=()*)

Bases: *Oper*

async **getLiftHints**(*runt*, *path*)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**FiniBlock**(*astinfo*, *kids=()*)

Bases: *AstNode*

An AST node that runs only once after all nodes have been consumed.

Example

Using a fini block:

```
fini {
   // stuff here runs *once* after the last node yield (even if there are no nodes)
}
```

Notes

A fini block must be runtsafe.

async run(*runt*, *genr*)

class synapse.lib.ast.**ForLoop**(*astinfo*, *kids=()*)

Bases: *Oper*

getRuntVars(*runt*)

async run(*runt*, *genr*)

class synapse.lib.ast.**FormName**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

class synapse.lib.ast.**FormPivot**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

-> foo:bar

buildgenr(*runt*, *name*)

pivogenr(*runt*, *prop*)

async run(*runt*, *genr*)

class synapse.lib.ast.**FormTagProp**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

class synapse.lib.ast.**FormatString**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

prepare()

class synapse.lib.ast.**FuncArgs**(*astinfo*, *kids=()*)

Bases: *AstNode*

Represents the function arguments in a function definition

async compute(*runt*, *path*)

class synapse.lib.ast.**FuncCall**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

class synapse.lib.ast.**Function**(*astinfo*, *kids=()*)

Bases: *AstNode*

(name, args, body)

// use args/kwargs syntax function bar(x, v=$(30)) { }

we auto-detect the behavior of the target function

return a value function bar(x, y) { return ($(x + y)) }

a function that produces nodes function bar(x, y) { [baz:faz=(x, y)] }

$foo = $bar(10, v=20)

async callfunc(*runt*, *argdefs*, *args*, *kwargs*)

Execute a function call using the given runtime.

This function may return a value / generator / async generator

getRuntVars(*runt*)

isRuntSafe(*runt*)

prepare()

async run(*runt*, *genr*)

runtopaque = True

validate(*runt*)

class synapse.lib.ast.**HasAbsPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**HasRelPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

async getLiftHints(*runt*, *path*)

async hasProp(*node*, *runt*, *name*)

class synapse.lib.ast.**HasTagPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**IfClause**(*astinfo*, *kids=()*)

Bases: *AstNode*

class synapse.lib.ast.**IfStmt**(*astinfo*, *kids=()*)

Bases: *Oper*

prepare()

async run(*runt*, *genr*)

class synapse.lib.ast.**InitBlock**(*astinfo*, *kids=()*)

Bases: *AstNode*

An AST node that runs only once before yielding nodes.

Example

Using a init block:

```
init {
    // stuff here runs *once* before the first node yield (even if there are no
↪nodes)
}
```

async run(*runt*, *genr*)

class synapse.lib.ast.**LiftByArray**(*astinfo*, *kids=()*)

Bases: *LiftOper*

:prop*[range=(200, 400)]

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftFormTag**(*astinfo*, *kids=()*)

Bases: *LiftOper*

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftFormTagProp**(*astinfo*, *kids=()*)

Bases: *LiftOper*

hehe:haha#foo.bar:baz [= x]

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftOper**(*astinfo*, *kids=()*)

Bases: *Oper*

async lift(*runt*, *path*)

reverseLift(*astinfo*)

async run(*runt*, *genr*)

class synapse.lib.ast.**LiftProp**(*astinfo*, *kids=()*)

Bases: *LiftOper*

async getRightHints(*runt*, *path*)

async lift(*runt*, *path*)

async proplift(*prop*, *runt*, *path*)

class synapse.lib.ast.**LiftPropBy**(*astinfo*, *kids=()*)

Bases: *LiftOper*

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftTag**(*astinfo*, *kids=()*)

Bases: *LiftOper*

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftTagProp**(*astinfo*, *kids=()*)

Bases: *LiftOper*

#foo.bar:baz [= x]

async lift(*runt*, *path*)

class synapse.lib.ast.**LiftTagTag**(*astinfo*, *kids=()*)

Bases: *LiftOper*

##foo.bar

async lift(*runt*, *path*)

class synapse.lib.ast.**List**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

prepare()

repr()

class synapse.lib.ast.**LookList**(*astinfo*, *kids=()*)

Bases: *AstNode*

class synapse.lib.ast.**Lookup**(*astinfo*, *kids*, *autoadd=False*)

Bases: *Query*

When storm input mode is "lookup"

async run(*runt*, *genr*)

class synapse.lib.ast.**N1Walk**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *Oper*

buildfilter(*runt*, *destforms*, *cmpr*)

repr()

async run(*runt*, *genr*)

async walkNodeEdges(*runt*, *node*, *verb=None*)

class synapse.lib.ast.**N1WalkNPivo**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOut*

async run(*runt*, *genr*)

class synapse.lib.ast.**N2Walk**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *N1Walk*

async walkNodeEdges(*runt*, *node*, *verb=None*)

class synapse.lib.ast.**N2WalkNPivo**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotIn*

async run(*runt*, *genr*)

class synapse.lib.ast.**NotCond**(*astinfo*, *kids=()*)

Bases: *Cond*

not <cond>

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**Oper**(*astinfo*, *kids=()*)

Bases: *AstNode*

class synapse.lib.ast.**OrCond**(*astinfo*, *kids=()*)

Bases: *Cond*

<cond> or <cond>

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**PivotIn**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

<- *

async getPivsIn(*runt*, *node*, *path*)

async run(*runt*, *genr*)

class synapse.lib.ast.**PivotInFrom**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

<- foo:edge

async run(*runt*, *genr*)

class synapse.lib.ast.**PivotOper**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *Oper*

repr()

class synapse.lib.ast.**PivotOut**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

-> *

async getPivsOut(*runt*, *node*, *path*)

async run(*runt*, *genr*)

class synapse.lib.ast.**PivotToTags**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

-> # pivot to all leaf tag nodes -> #* pivot to all tag nodes -> #cno.* pivot to all tag nodes which match cno.* -> #foo.bar pivot to the tag node foo.bar if present

async run(*runt*, *genr*)

class synapse.lib.ast.**PropName**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

prepare()

class synapse.lib.ast.**PropPivot**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

:foo -> bar:foo

buildgenr(*runt*, *name*)

pivogenr(*runt*, *prop*)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**PropPivotOut**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

:prop -> *

async **run**(*runt*, *genr*)

class synapse.lib.ast.**PropValue**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

async **getPropAndValu**(*runt*, *path*)

isRuntSafe(*runt*)

isRuntSafeAtom(*runt*)

prepare()

class synapse.lib.ast.**Query**(*astinfo*, *kids=()*)

Bases: *AstNode*

async **iterNodePaths**(*runt*, *genr=None*)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**RawPivot**(*astinfo*, *kids=()*, *isjoin=False*)

Bases: *PivotOper*

-> { <varsfrompath> }

async **run**(*runt*, *genr*)

class synapse.lib.ast.**RelProp**(*astinfo*, *kids=()*)

Bases: *PropName*

class synapse.lib.ast.**RelPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

(:foo:bar or .univ) <cmpr> <value>

async **getCondEval**(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

async **getLiftHints**(*runt*, *path*)

class synapse.lib.ast.**RelPropValue**(*astinfo*, *kids=()*)

Bases: *PropValue*

class synapse.lib.ast.**Return**(*astinfo*, *kids=()*)

Bases: *Oper*

async **run**(*runt*, *genr*)

class synapse.lib.ast.**Search**(*astinfo*, *kids=()*)

Bases: *Query*

async **run**(*runt*, *genr*)

class synapse.lib.ast.**SetItemOper**(*astinfo*, *kids=()*)

Bases: *Oper*

$foo.bar = baz $foo."bar baz" = faz $foo.$bar = baz

async **run**(*runt*, *genr*)

class synapse.lib.ast.**SetVarOper**(*astinfo*, *kids=()*)

Bases: *Oper*

getRuntVars(*runt*)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**Stop**(*astinfo*, *kids=()*)

Bases: *Oper*

async **run**(*runt*, *genr*)

class synapse.lib.ast.**SubGraph**(*rules*)

Bases: object

An Oper like object which generates a subgraph.

Notes

The rules format for the subgraph is shaped like the following:

```
rules = {

    'degrees': 1,

    'edges': True,
    'edgelimit': 3000,
    'filterinput': True,
    'yieldfiltered': False,

    'filters': [
        '-(#foo or #bar)',
        '-(foo:bar or baz:faz)',
    ],

```

(continues on next page)

(continued from previous page)

```
    'pivots': [
        '-> * | limit 100',
        '<- * | limit 100',
    ]

    'forms': {

        'inet:fqdn':{
            'filters': [],
            'pivots': [],
        }

        '*': {
            'filters': [],
            'pivots': [],
        },
    },
}
```

Nodes which were original seeds have path.meta('graph:seed').

All nodes have path.meta('edges') which is a list of (iden, info) tuples.

async omit(*runt*, *node*)

async pivots(*runt*, *node*, *path*, *existing*)

async run(*runt*, *genr*)

class synapse.lib.ast.**SubQuery**(*astinfo*, *kids=()*)

Bases: *Oper*

async compute(*runt*, *path*)

Use subquery as a value. It is error if the subquery used in this way doesn't yield exactly one node or has a return statement.

Its value is the primary property of the node yielded, or the returned value.

async compute_array(*runt*, *path*)

Use subquery as an array.

async inline(*runt*, *genr*)

Operate subquery as if it were inlined

isRuntSafe(*runt*)

async run(*runt*, *genr*)

class synapse.lib.ast.**SubqCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async getCondEval(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**SwitchCase**(*astinfo*, *kids=()*)

Bases: *Oper*

prepare()

async **run**(*runt*, *genr*)

class synapse.lib.ast.**TagCond**(*astinfo*, *kids=()*)

Bases: *Cond*

#foo.bar

async **getCondEval**(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

async **getLiftHints**(*runt*, *path*)

class synapse.lib.ast.**TagMatch**(*astinfo*, *kids=()*)

Bases: *TagName*

Like TagName, but can have asterisks

async **compute**(*runt*, *path*)

hasglob()

class synapse.lib.ast.**TagName**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

async **computeTagArray**(*runt*, *path*, *excignore=()*)

prepare()

class synapse.lib.ast.**TagProp**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**TagPropCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async **getCondEval**(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**TagPropValue**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**TagValuCond**(*astinfo*, *kids=()*)

Bases: *Cond*

async **getCondEval**(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

class synapse.lib.ast.**TagValue**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

isRuntSafe(*runt*)

isRuntSafeAtom(*runt*)

class synapse.lib.ast.**TryCatch**(*astinfo*, *kids=()*)

Bases: *AstNode*

async **getCatchBlock**(*name*, *runt*, *path=None*)

async **getErrValu**(*e*)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**UnaryExprNode**(*astinfo*, *kids=()*)

Bases: *Value*

A unary (i.e. single-argument) expression node

async **compute**(*runt*, *path*)

prepare()

class synapse.lib.ast.**UnivProp**(*astinfo*, *kids=()*)

Bases: *RelProp*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**UnivPropValue**(*astinfo*, *kids=()*)

Bases: *PropValue*

class synapse.lib.ast.**Value**(*astinfo*, *kids=()*)

Bases: *AstNode*

The base class for all values and value expressions.

async **compute**(*runt*, *path*)

async **getCondEval**(*runt*)

Return a function that may be used to evaluate the boolean truth of the value expression using a runtime and optional node path.

async **getLiftHints**(*runt*, *path*)

isRuntSafe(*runt*)

class synapse.lib.ast.**VarDeref**(*astinfo*, *kids=()*)

Bases: *Value*

async **compute**(*runt*, *path*)

class synapse.lib.ast.**VarEvalOper**(*astinfo*, *kids=()*)

Bases: *Oper*

Facilitate a stand-alone operator that evaluates a var. $foo.bar(“baz”)

async **run**(*runt*, *genr*)

class synapse.lib.ast.**VarList**(*astinfo*, *valu*, *kids=()*)

Bases: *Const*

class synapse.lib.ast.**VarListSetOper**(*astinfo*, *kids=()*)

Bases: *Oper*

getRuntVars(*runt*)

async run(*runt*, *genr*)

class synapse.lib.ast.**VarValue**(*astinfo*, *kids=()*)

Bases: *Value*

async compute(*runt*, *path*)

hasVarName(*name*)

isRuntSafe(*runt*)

isRuntSafeAtom(*runt*)

prepare()

validate(*runt*)

class synapse.lib.ast.**WhileLoop**(*astinfo*, *kids=()*)

Bases: *Oper*

async run(*runt*, *genr*)

class synapse.lib.ast.**YieldValu**(*astinfo*, *kids=()*)

Bases: *Oper*

async run(*runt*, *genr*)

async yieldFromValu(*runt*, *valu*)

async synapse.lib.ast.**expr_add**(*x*, *y*)

async synapse.lib.ast.**expr_div**(*x*, *y*)

async synapse.lib.ast.**expr_eq**(*x*, *y*)

async synapse.lib.ast.**expr_ge**(*x*, *y*)

async synapse.lib.ast.**expr_gt**(*x*, *y*)

async synapse.lib.ast.**expr_le**(*x*, *y*)

async synapse.lib.ast.**expr_lt**(*x*, *y*)

async synapse.lib.ast.**expr_mod**(*x*, *y*)

async synapse.lib.ast.**expr_mul**(*x*, *y*)

async synapse.lib.ast.**expr_ne**(*x*, *y*)

async synapse.lib.ast.**expr_neg**(*x*)

async synapse.lib.ast.**expr_not**(*x*)

async synapse.lib.ast.**expr_pow**(*x*, *y*) async synapse.lib.ast.**expr_prefix**(*x*, *y*)

async synapse.lib.ast.**expr_re**(*x*, *y*)

async synapse.lib.ast.**expr_sub**(*x*, *y*)

synapse.lib.ast.**parseNumber**(*x*)

async synapse.lib.ast.**pullone**(*genr*)

synapse.lib.autodoc module

class synapse.lib.autodoc.**RstHelp**

Bases: `object`

addHead(*name*, *lvl=0*, *link=None*, *addprefixline=True*, *addsuffixline=True*)

addLines(**lines*)

getRstText()

synapse.lib.autodoc.**docStormTypes**(*page*, *docinfo*, *linkprefix*, *islib=False*, *lvl=1*, *known_types=None*, *types_prefix=None*, *types_suffix=None*)

Process a list of StormTypes doc information to add them to an RstHelp object.

Notes

This will create internal hyperlink link targets for each header item. The link prefix string must be given with the `linkprefix` argument.

Parameters

- **page** (`RstHelp`) – The RST page to add .
- **docinfo** (*dict*) – A Stormtypes Doc.
- **linkprefix** (*str*) – The RST link prefix string to use.
- **islib** (*bool*) – Treat the data as a library. This will preface the header and attribute values with $ and use full paths for attributes.
- **lvl** (*int*) – The base header level to use when adding headers to the page.

Returns

None

synapse.lib.autodoc.**genCallsig**(*rtype*)

synapse.lib.autodoc.**getArgLines**(*rtype*)

synapse.lib.autodoc.**getLink**(*sname*, *linkprefix*, *ref=False*, *suffix=None*)

synapse.lib.autodoc.**getReturnLines**(*rtype*, *known_types=None*, *types_prefix=None*, *suffix=None*, *isstor=False*)

synapse.lib.autodoc.**getRtypeStr**(*rtype*, *known_types*, *types_prefix*, *suffix*)

synapse.lib.autodoc.**ljuster**(*ilines*)

Helper to lstrip lines of whitespace an appropriate amount.

`synapse.lib.autodoc.prepareRstLines(doc)`

Prepare a desc string for RST lines.

`synapse.lib.autodoc.runtimeDocStormTypes(page, docinfo, islib=False, lvl=1, oneline=False, addheader=True, preamble=None)`

Process a list of StormTypes doc information to add them to a RstHelp object.

Used for Storm runtime help generation.

Parameters

- **page** (`RstHelp`) – The RST page to add .
- **docinfo** (*dict*) – A Stormtypes Doc.
- **linkprefix** (*str*) – The RST link prefix string to use.
- **islib** (*bool*) – Treat the data as a library. This will preface the header and attribute values with $ and use full paths for attributes.
- **lvl** (*int*) – The base header level to use when adding headers to the page.
- **oneline** (*bool*) – Only display the first line of description. Omits local headers.
- **preamble** (*list*) – Lines added after the header; and before locls.

Returns

None

`synapse.lib.autodoc.runtimeGetArgLines(rtype)`

`synapse.lib.autodoc.runtimeGetReturnLines(rtype, isstor=False)`

`synapse.lib.autodoc.scrubLines(lines)`

Remove any empty lines until we encounter non-empty linee

synapse.lib.base module

class `synapse.lib.base.Base`

Bases: `object`

Base class for Synapse objects.

Acts as an observable, enables async init and fini.

Example

class Foo(Base):

async def __anit__(self, x, y):

await Base.__anit__(self)

await stuff(x, y)

foo = await Foo.anit(10)

Note: One should not create instances directly via its initializer, i.e. Base(). One shall always use the class method anit.

`async addSignalHandlers()`

Register SIGTERM/SIGINT signal handlers with the ioloop to fini this object.

`async classmethod anit(*args, **kwargs)`

`async dist(mesg)`

Distribute an existing event tuple.

Parameters

mesg (*(str,dict)*) – An event tuple.

Example

await base.dist(('foo',{'bar':'baz'}))

`async enter_context(item)`

Modeled on Python's contextlib.ExitStack.enter_context. Enters a new context manager and adds its __exit__() and __aexit__ method to its onfini handlers.

Returns

The result of item's own __aenter__ or __enter__() method.

`async fini()`

Shut down the object and notify any onfini() coroutines.

Returns

Remaining ref count

`async fire(evtname, **info)`

Fire the given event name on the Base. Returns a list of the return values of each callback.

Example

for ret in d.fire('woot',foo='asdf'):
print('got: %r' % (ret,))

`incref()`

Increment the reference count for this base. This API may be optionally used to control fini().

`link(func)`

Add a callback function to receive *all* events.

Example

base1 = Base() base2 = Base()

base1.link(base2.dist)

all events on base1 are also propagated on base2

`async main()`

Helper function to setup signal handlers for this base as the main object. (use base.waitfini() to block)

Note: This API may only be used when the ioloop is *also* the main thread.

`off`(*evnt*, *func*)

Remove a previously registered event handler function.

Example

base.off('foo', onFooFunc)

`on`(*evnt*, *func*, *base=None*)

Add an base function callback for a specific event with optional filtering. If the function returns a coroutine, it will be awaited.

Parameters

- **evnt** (*str*) – An event name
- **func** (*function*) – A callback function to receive event tufo

Examples

Add a callback function and fire it:

async def baz(event):
x = event[1].get('x') y = event[1].get('y') return x + y

d.on('foo', baz)

this fire triggers baz… await d.fire('foo', x=10, y=20)

Return type
None

`onWith`(*evnt*, *func*)

A context manager which can be used to add a callback and remove it when using a `with` statement.

Parameters

- **evnt** (*str*) – An event name
- **func** (*function*) – A callback function to receive event tufo

`onfini`(*func*)

Add a function/coroutine/Base to be called on fini().

`async postAnit`()

Method called after self.__anit__() has completed, but before anit() returns the object to the caller.

`schedCallSafe`(*func*, **args*, ***kwargs*)

Schedule a function to run as soon as possible on the same event loop that this Base is running on.

This function does *not* pend on the function completion.

Parameters

- **func** –
- ***args** –
- ****kwargs** –

Notes

This method may be called from outside of the event loop on a different thread. This function will break any task scoping done with synapse.lib.scope.

Returns
A Future representing the eventual function execution.

Return type
concurrent.futures.Future

`schedCoro`(*coro*)

Schedules a free-running coroutine to run on this base's event loop. Kills the coroutine if Base is fini'd. It does not pend on coroutine completion.

Parameters
coro – The coroutine to schedule.

Notes

This function is *not* threadsafe and must be run on the Base's event loop. Tasks created by this function do inherit the synapse.lib.scope Scope from the current task.

Returns
An asyncio.Task object.

Return type
asyncio.Task

`schedCoroSafe`(*coro*)

Schedules a coroutine to run as soon as possible on the same event loop that this Base is running on.

This function does *not* pend on coroutine completion.

Notes

This method may be run outside the event loop on a different thread. This function will break any task scoping done with synapse.lib.scope.

Returns
A Future representing the eventual coroutine execution.

Return type
concurrent.futures.Future

`schedCoroSafePend`(*coro*)

Schedules a coroutine to run as soon as possible on the same event loop that this Base is running on

Note: This method may *not* be run inside an event loop

`unlink`(*func*)

Remove a callback function previously added with link()

Example

base.unlink(callback)

`waiter`(*count*, **names*)

Construct and return a new Waiter for events on this base.

Example

wait up to 3 seconds for 10 foo:bar events…

waiter = base.waiter(10,'foo:bar')

.. fire task that will cause foo:bar events

events = await waiter.wait(timeout=3)

if events == None:
  # handle the timeout case…

for event in events:
  # parse the events if you need…

Note: Use this with caution. It's easy to accidentally construct race conditions with this mechanism ;)

`async waitfini`(*timeout=None*)

Wait for the base to fini()

Returns
None if timed out, True if fini happened

Example

base.waitfini(timeout=30)

`class synapse.lib.base.BaseRef`

Bases: `Base`

An object for managing multiple Base instances by name.

`async gen`(*name*)

Atomically get/gen a Base and incref. (requires ctor during BaseRef init)

Parameters
`name` (*str*) – The name/iden of the Base instance.

`get`(*name*)

Retrieve a Base instance by name.

Parameters
`name` (*str*) – The name/iden of the Base

Returns
The Base instance (or None)

Return type
(*Base*)

items()

pop(*name*)

Remove and return a Base from the BaseRef.

Parameters
name (*str*) – The name/iden of the Base instance

Returns
The named base (or None)

Return type
(*Base*)

put(*name*, *base*)

Add a Base (or sub-class) to the BaseRef by name.

Parameters

- **name** (*str*) – The name/iden of the Base
- **base** (Base) – The Base instance

Returns
(None)

vals()

class synapse.lib.base.**Waiter**(*base*, *count*, **names*)

Bases: object

A helper to wait for a given number of events on a Base.

fini()

async **wait**(*timeout=None*)

Wait for the required number of events and return them or None on timeout.

Example

evnts = waiter.wait(timeout=30)

if evnts == None:
handleTimedOut() return

for evnt in evnts:
doStuff(evnt)

async synapse.lib.base.**main**(*coro*)

async synapse.lib.base.**schedGenr**(*genr*, *maxsize=100*)

Schedule a generator to run on a separate task and yield results to this task (pipelined generator).

synapse.lib.boss module

class synapse.lib.boss.**Boss**

Bases: *Base*

An object to track "promoted" async tasks.

async execute(*coro*, *name*, *user*, *info=None*, *iden=None*)

Create a synapse task from the given coroutine.

get(*iden*)

async promote(*name*, *user*, *info=None*, *taskiden=None*)

Promote the currently running task.

Parameters

- **name** (*str*) – The name of the task.
- **user** – The User who owns the task.
- **taskiden** – An optional GUID for the task.
- **info** – An optional information dictionary containing information about the task.

Returns

The Synapse Task object.

Return type

s_task.Task

async promotetask(*task*, *name*, *user*, *info=None*, *taskiden=None*)

ps()

synapse.lib.cache module

A few speed optimized (lockless) cache helpers. Use carefully.

class synapse.lib.cache.**EdgeGlobs**

Bases: *TagGlobs*

class synapse.lib.cache.**FixedCache**(*callback*, *size=10000*)

Bases: object

async aget(*key*)

clear()

get(*key*)

pop(*key*)

put(*key*, *val*)

class synapse.lib.cache.**LruDict**(*size=10000*)

Bases: MutableMapping

Maintains the last n accessed keys

get(*key*, *default=None*)

Note: we override default impl from parent to avoid costly KeyError

items() → a set-like object providing a view on D's items

values() → an object providing a view on D's values

class synapse.lib.cache.**TagGlobs**

Bases: object

An object that manages multiple tag globs and values for caching.

add(*name*, *valu*, *base=None*)

get(*name*)

rem(*name*, *valu*)

synapse.lib.cache.**getEdgeGlobRegx**(*name*)

synapse.lib.cache.**getTagGlobRegx**(*name*)

synapse.lib.cache.**memoize**(*size=16384*)

synapse.lib.cache.**memoizemethod**(*size=16384*)

A version of memoize that doesn't cause GC cycles when applied to a method.

synapse.lib.cache.**regexizeEdgeGlob**(*edge*)

synapse.lib.cache.**regexizeTagGlob**(*tag*)

Returns

a regular expression string with ** and * interpreted as tag globs

Precondition:

tag is a valid tagmatch

Notes

A single asterisk will replace exactly one dot-delimited component of a tag A double asterisk will replace one or more of any character.

The returned string does not contain a starting '^' or trailing '$'.

synapse.lib.cell module

class synapse.lib.cell.**Cell**

Bases: *Pusher*, *Aware*

A Cell() implements a synapse micro-service.

A Cell has 5 phases of startup:

1. Universal cell data structures
2. Service specific storage/data (pre-nexs)
3. Nexus subsystem initialization

4. Service specific startup (with nexus)
5. Networking and mirror services

BACKUP_SPAWN_TIMEOUT = 60.0

COMMIT = ''

FREE_SPACE_CHECK_FREQ = 60.0

VERSION = (2, 167, 0)

VERSTRING = '2.167.0'

addActiveCoro(*func*, *iden=None*, *base=None*)

Add a function callback to be run as a coroutine when the Cell is active.

Parameters

- **func** (*coroutine function*) – The function run as a coroutine.
- **iden** (*str*) – The iden to use for the coroutine.
- **base** (*Optional[Base]*) – if present, this active coro will be fini'd when the base is fini'd

Returns

A GUID string that identifies the coroutine for delActiveCoro()

Return type

str

Note: This will re-fire the coroutine if it exits and the Cell is still active.

addHealthFunc(*func*)

Register a callback function to get a HealthCheck object.

addHttpApi(*path*, *ctor*, *info*)

async **addHttpSess**(*iden*, *info*)

async **addHttpsPort**(*port*, *host='0.0.0.0'*, *sslctx=None*)

Add a HTTPS listener to the Cell.

Parameters

- **port** (*int*) – The listening port to bind.
- **host** (*str*) – The listening host.
- **sslctx** (*ssl.SSLContext*) – An externally managed SSL Context.

Note: If the SSL context is not provided, the Cell will assume it manages the SSL context it creates for a given listener and will add a reload handler named https:certs to enabled reloading the SSL certificates from disk.

addReloadableSystem(*name: str*, *func: callable*)

Add a reloadable system. This may be dynamically called at at time.

Parameters

- **name** (*str*) – Name of the system.
- **func** – The callable for reloading a given system.

Note: Reload functions take no arguments when they are executed. Values returned by the reload function must be msgpack friendly.

Returns
None

async **addRole**(*name*)

async **addRoleRule**(*iden*, *rule*, *indx=None*, *gateiden=None*)

async **addSignalHandlers**()

Register SIGTERM/SIGINT signal handlers with the ioloop to fini this object.

async **addUser**(*name*, *passwd=None*, *email=None*, *iden=None*)

async **addUserApiKey**(*useriden*, *name*, *duration=None*)

Add an API key for a user.

Notes

The secret API key is only available once.

Parameters

- **useriden** (*str*) – User iden value.
- **name** (*str*) – Name of the API key.
- **duration** (*int or None*) – Duration of time for the API key to be valid (in milliseconds).

Returns
A tuple of the secret API key value and the API key metadata information.

Return type
tuple

async **addUserRole**(*useriden*, *roleiden*, *indx=None*)

async **addUserRule**(*iden*, *rule*, *indx=None*, *gateiden=None*)

async **behold**()

beholder()

cellapi

alias of *CellApi*

checkFreeSpace()

async **checkUserApiKey**(*key*)

Check if a user API key is valid.

Notes

If the key is not valid, the dictionary will contain a `mesg` key. If the key is valid, the dictionary will contain the user def in a `udef` key, and the key metadata in a `kdef` key.

Parameters
  key (*str*) – The API key to check.

Returns
  Tuple of two items, a boolean if the key is valid and a dictionary.

Return type
  tuple

confbase = {'_log_conf': {'description': 'Opaque structure used for logging by spawned processes.', 'hideconf': True, 'type': 'object'}, 'aha:admin': {'description': 'An AHA client certificate CN to register as a local admin user.', 'type': 'string'}, 'aha:leader': {'description': 'The AHA service name to claim as the active instance of a storm service.', 'type': 'string'}, 'aha:name': {'description': 'The name of the cell service in the aha service registry.', 'type': 'string'}, 'aha:network': {'description': 'The AHA service network. This makes aha:name/aha:leader relative names.', 'type': 'string'}, 'aha:provision': {'description': 'The telepath URL of the aha provisioning service.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:registry': {'description': 'The telepath URL of the aha service registry.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:svcinfo': {'description': 'An AHA svcinfo object. If set, this overrides self discovered Aha service information.', 'hidecmdl': True, 'hidedocs': True, 'properties': {'urlinfo': {'properties': {'host': {'type': 'string'}, 'port': {'type': 'integer'}, 'schema': {'type': 'string'}}, 'required': ('host', 'port', 'scheme'), 'type': 'object'}}, 'required': ('urlinfo',), 'type': 'object'}, 'aha:user': {'description': 'The username of this service when connecting to others.', 'type': 'string'}, 'auth:anon': {'description': 'Allow anonymous telepath access by mapping to the given user name.', 'type': 'string'}, 'auth:conf': {'description': 'Extended configuration to be used by an alternate auth constructor.', 'hideconf': True, 'type': 'object'}, 'auth:ctor': {'description': 'Allow the construction of the cell auth object to be hooked at runtime.', 'hideconf': True, 'type': 'string'}, 'auth:passwd': {'description': 'Set to <passwd> (local only) to bootstrap the root user password.', 'type': 'string'}, 'backup:dir': {'description': 'A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.', 'type': 'string'}, 'cell:ctor': {'description': 'An optional python path to the Cell class. Used by stemcell.', 'hideconf': True, 'type': 'string'}, 'cell:guid': {'description': 'An optional hard-coded GUID to store as the permanent GUID for the service.', 'hideconf': True, 'type': 'string'}, 'dmon:listen': {'description': 'A config-driven way to specify the telepath bind URL.', 'type': ['string', 'null']}, 'https:headers': {'description': 'Headers to add to all HTTPS server responses.', 'hidecmdl': True, 'type': 'object'}, 'https:parse:proxy:remoteip': {'default': False, 'description': 'Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.', 'type': 'boolean'}, 'https:port': {'description': 'A config-driven way to specify the HTTPS port.', 'type': ['integer', 'null']}, 'inaugural': {'description': 'Data used to drive configuration of the service upon first startup.', 'hidedocs': True, 'properties': {'roles': {'items': {'additionalProperties': False, 'properties': {'name': {'pattern': '^(?!all$).+$', 'type': 'string'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}, 'users': {'items': {'additionalProperties': False, 'properties': {'admin': {'default': False, 'type': 'boolean'}, 'email': {'type': 'string'}, 'name': {'pattern': '^(?!root$).+$', 'type': 'string'}, 'roles': {'items': {'type': 'string'}, 'type': 'array'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}}, 'type': 'object'}, 'limit:disk:free': {'default': 5, 'description': 'Minimum disk free space percentage before setting the cell read-only.', 'maximum': 100, 'minimum': 0, 'type': ['integer', 'null']}, 'max:users': {'default': 0, 'description': 'Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).', 'minimum': 0, 'type': 'integer'}, 'mirror': {'description': 'A telepath URL for our upstream mirror (we must be a backup!).', 'hidecmdl': True, 'hidedocs': True, 'type': ['string', 'null']}, 'nexslog:async': {'default': True, 'description': 'Set to false to disable async memory mapping of the nexus change log.', 'hidecmdl': True, 'hidedocs': True, 'type': 'boolean'}, 'nexslog:en': {'default': False, 'description': 'Record all changes to a stream

confdefs = {}

async **cullNexsLog**(*offs*)

async **delActiveCoro**(*iden*)

Remove an Active coroutine previously added with addActiveCoro().

Parameters

iden (*str*) – The iden returned by addActiveCoro()

async **delBackup**(*name*)

async **delHttpSess**(*iden*)

async **delRole**(*iden*)

async **delRoleRule**(*iden*, *rule*, *gateiden=None*)

async **delUser**(*iden*)

async **delUserApiKey**(*iden*)

Delete an existing API key.

Parameters

iden (*str*) – The iden of the API key to delete.

Returns

True indicating the key was deleted.

Return type

bool

async **delUserRole**(*useriden*, *roleiden*)

async **delUserRule**(*iden*, *rule*, *gateiden=None*)

async **dyncall**(*iden*, *todo*, *gatekeys=()*)

async **dyniter**(*iden*, *todo*, *gatekeys=()*)

async classmethod **execmain**(*argv*, *outp=None*)

The main entry point for running the Cell as an application.

Parameters

- **argv** (*list*) – A list of command line arguments to launch the Cell with.
- **outp** (*s_ouput.OutPut*) – Optional, an output object. No longer used in the default implementation.

Notes

This coroutine waits until the Cell is fini'd or a SIGINT/SIGTERM signal is sent to the process.

Returns
None.

`async feedBeholder(name, info, gates=None, perms=None)`

Feed a named event onto the `cell:beholder` message bus that will sent to any listeners.

Parameters

- **info** (*dict*) – An information dictionary to be sent to any consumers.
- **gates** (*list*) – List of gate idens, whose details will be added to the outbound message(s).
- **perms** (*list*) – List of permission names, whose details will be added to the outbound message(s).

Returns
None

`async fini()`

Fini override that ensures locking teardown order.

`async genHttpSess(iden)`

`async genUserOnepass(iden, duration=600000)`

`async getAhaInfo()`

`async getApiKeys()`

Get all API keys in the cell.

Yields
tuple – kdef values

`classmethod getArgParser(conf=None)`

Get an `argparse.ArgumentParser` for the Cell.

Parameters
conf (*s_config.Config*) – Optional, a Config object which

Notes

Boot time configuration data is placed in the argument group called `config`. This adds default `dirn`, `--telepath`, `--https` and `--name` arguements to the argparser instance. Configuration values which have the `hideconf` or `hidecmdl` value set to True are not added to the argparser instance.

Returns
A ArgumentParser for the Cell.

Return type
argparse.ArgumentParser

`async getAuthGate(iden)`

`async getAuthGates()`

`async getAuthRoles()`

async **getAuthUsers**(*archived=False*)

async **getBackupInfo**()

Gets information about recent backup activity

async **getBackups**()

getCachedSslCtx(*opts=None*, *verify=None*)

async **getCellApi**(*link*, *user*, *path*)

Get an instance of the telepath Client object for a given user, link and path.

Parameters

- **link** (*s_link.Link*) – The link object.
- **user** (*s_hive.HiveUser*) – The heavy user object.
- **path** (*str*) – The path requested.

Notes

This defaults to the self.cellapi class. Implementors may override the default class attribute for cellapi to share a different interface.

Returns
The shared object for this cell.

Return type
object

getCellIden()

async **getCellInfo**()

Return metadata specific for the Cell.

Notes

By default, this function returns information about the base Cell implementation, which reflects the base information in the Synapse Cell.

It is expected that implementers override the following Class attributes in order to provide meaningful version information:

`COMMIT` - A Git Commit `VERSION` - A Version tuple. `VERSTRING` - A Version string.

Returns
A Dictionary of metadata.

Return type
Dict

getCellNexsRoot()

async **getCellRunId**()

classmethod **getCellType**()

async **getConfOpt**(*name*)

async **getDmonSessions**()

getDmonUser()

Return the user IDEN of a telepath caller who invoked the current task. (defaults to returning current root user)

classmethod **getEnvPrefix**()

Get a list of envar prefixes for config resolution.

async **getHealthCheck**()

async **getHiveKey**(*path*)

Get the value of a key in the cell default hive

async **getHiveKeys**(*path*)

Return a list of (name, value) tuples for nodes under the path.

async **getHttpSessDict**(*iden*)

getLocalProxy(*share='*'*, *user='root'*)

getLocalUrl(*share='*'*, *user='root'*)

async **getLogExtra**(***kwargs*)

Get an extra dictionary for structured logging which can be used as a extra argument for loggers.

Parameters

`**kwargs` – Additional key/value items to add to the log.

Returns

A dictionary

Return type

Dict

async **getMirrorUrls**()

async **getNexsIndx**()

async **getNexusChanges**(*offs*, *tellready=False*)

getPermDef(*perm*)

getPermDefs()

getReloadableSystems()

async **getRoleDef**(*iden*)

async **getRoleDefByName**(*name*)

async **getRoleDefs**()

getSpooledSet()

async **getSystemInfo**()

Get info about the system in which the cell is running

Returns

- volsize - Volume where cell is running total space

- volfree - Volume where cell is running free space
- backupvolsize - Backup directory volume total space
- backupvolfree - Backup directory volume free space
- cellstarttime - Cell start time in epoch milliseconds
- celluptime - Cell uptime in milliseconds
- cellrealdisk - Cell's use of disk, equivalent to du
- cellapprdisk - Cell's apparent use of disk, equivalent to ls -l
- osversion - OS version/architecture
- pyversion - Python version
- totalmem - Total memory in the system
- availmem - Available memory in the system
- cpucount - Number of CPUs on system
- tmpdir - The temporary directory interpreted by the Python runtime.

Return type
A dictionary with the following keys. All size values are in bytes

`async getTeleApi(`*link*, *mesg*, *path*`)`

Return a shared object for this link. :param link: A network link. :type link: synapse.lib.link.Link :param mesg: The tele:syn handshake message. :type mesg: (str,dict)

`getTempDir()`

`async getUserApiKey(`*iden*`)`

Get a user API key via iden.

Notes

This contains the raw value. Callers are responsible for removing the `shadow` key.

Parameters
`iden` (*str*) – The key iden.

Returns
The key dictionary; or none.

Return type
dict

`async getUserDef(`*iden*, *packroles=True*`)`

`async getUserDefByName(`*name*`)`

`async getUserDefs()`

`getUserName(`*iden*, *defv='<unknown>'*`)`

Translate the user iden to a user name.

`async getUserProfInfo(`*iden*, *name*`)`

`async getUserProfile(`*iden*`)`

async **getUserVarValu**(*iden*, *name*)

async **handoff**(*turl*, *timeout=30*)

Hand off leadership to a mirror in a transactional fashion.

async **hasHttpSess**(*iden*)

classmethod **initCellConf**(*conf=None*)

Create a Config object for the Cell.

Parameters

conf (*s_config.Config*) – An optional config structure. This has _opts_data taken from it.

Notes

The Config object has a `envar_prefix` set according to the results of `cls.getEnvPrefix()`.

Returns

A Config helper object.

Return type

s_config.Config

async classmethod **initFromArgv**(*argv*, *outp=None*)

Cell launcher which does automatic argument parsing, environment variable resolution and Cell creation.

Parameters

- **argv** (*list*) – A list of command line arguments to launch the Cell with.
- **outp** (*s_ouput.OutPut*) – Optional, an output object. No longer used in the default implementation.

Notes

This does the following items:

- Create a Config object from the Cell class.
- Creates an Argument Parser from the Cell class and Config object.
- Parses the provided arguments.
- Loads configuration data from the parsed options and environment variables.
- Sets logging for the process.
- Creates the Cell from the Cell Ctor.
- Adds a Telepath listener, HTTPs port listeners and Telepath share names.
- Returns the Cell.

Returns

This returns an instance of the Cell.

Return type

Cell

async **initNexusSubsystem**()

async **initServiceActive**()

async **initServiceNetwork**()

async **initServicePassive**()

async **initServiceRuntime**()

async **initServiceStorage**()

initSslCtx(*certpath*, *keypath*)

isActiveCoro(*iden*)

async **isCellActive**()

async **isRoleAllowed**(*iden*, *perm*, *gateiden=None*)

async **isUserAllowed**(*iden*, *perm*, *gateiden=None*, *default=False*)

async **iterBackupArchive**(*name*, *user*)

async **iterNewBackupArchive**(*user*, *name=None*, *remove=False*)

async **iterUserVars**(*iden*)

async **kill**(*user*, *iden*)

async **listHiveKey**(*path=None*)

async **listUserApiKeys**(*useriden*)

Get all the API keys for a user.

Parameters
useriden (*str*) – The user iden.

Returns
A list of kdef values.

Return type
list

async **loadHiveTree**(*tree*, *path=()*, *trim=False*)

Note: this is for expert emergency use only.

modCellConf(*conf*)

Modify the Cell's ondisk configuration overrides file and runtime configuration.

Parameters
conf (*dict*) – A dictionary of items to set.

Notes

This does require the data being set to be schema valid.

Returns
None.

async **modUserApiKey**(*iden*, *key*, *valu*)

Update a value in the user API key metadata.

Parameters

- **iden** (*str*) – Iden of the key to update.
- **key** (*str*) – Name of the valu to update.
- **valu** – The new value.

Returns
An updated key metadata dictionary.

Return type
dict

popCellConf(*name*)

Remove a key from the Cell's ondisk configuration overrides file and runtime configuration.

Parameters
name (*str*) – Name of the value to remove.

Notes

This does **not** modify the cell.yaml file. This does re-validate the configuration after removing the value, so if the value removed had a default populated by schema, that default would be reset.

Returns
None

async **popHiveKey**(*path*)

Remove and return the value of a key in the cell default hive.

Note: this is for expert emergency use only.

async **popUserProfInfo**(*iden*, *name*, *default=None*)

async **popUserVarValu**(*iden*, *name*, *default=None*)

async **promote**(*graceful=False*)

Transform this cell from a passive follower to an active cell that writes changes locally.

async **ps**(*user*)

async **readyToMirror**()

async **reload**(*subsystem=None*)

async **reqAhaProxy**(*timeout=None*)

async **reqGateKeys**(*gatekeys*)

async **rotateNexsLog**()

`runActiveTask(coro)`

`async runBackup(name=None, wait=True)`

`async saveHiveTree(path=())`

`async setCellActive(active)`

`async setHiveKey(path, valu)`

Set or change the value of a key in the cell default hive

`async setHttpSessInfo(iden, name, valu)`

`async setNexsIndx(indx)`

`async setRoleName(iden, name)`

`async setRoleRules(iden, rules, gateiden=None)`

`async setUserAdmin(iden, admin, gateiden=None)`

`async setUserArchived(iden, archived)`

`async setUserEmail(useriden, email)`

`async setUserLocked(iden, locked)`

`async setUserName(useriden, name)`

`async setUserPasswd(iden, passwd)`

`async setUserProfInfo(iden, name, valu)`

`async setUserRoles(useriden, roleidens)`

`async setUserRules(iden, rules, gateiden=None)`

`async setUserVarValu(iden, name, valu)`

`async sync()`

no-op mutable for testing purposes. If I am follower, when this returns, I have received and applied all the writes that occurred on the leader before this call.

`async trimNexsLog(consumers=None, timeout=30)`

`async tryUserPasswd(name, passwd)`

`async updateHttpSessInfo(iden, vals: dict)`

`async waitNexsOffs(offs, timeout=None)`

`class synapse.lib.cell.CellApi`

Bases: `Base`

`addAuthRole(name)`

`addAuthRule(name, rule, indx=None, gateiden=None)`

This API is deprecated.

`addRole(name)`

addRoleRule(*iden*, *rule*, *indx=None*, *gateiden=None*)

addUser(*name*, *passwd=None*, *email=None*, *iden=None*)

addUserRole(*useriden*, *roleiden*, *indx=None*)

addUserRule(*iden*, *rule*, *indx=None*, *gateiden=None*)

async allowed(*perm*, *default=None*)

Check if the user has the requested permission.

Parameters

- **perm** – permission path components to check
- **default** – Value returned if no value stored

Examples

Form a path and check the permission from a remote proxy:

```
perm = ('node', 'add', 'inet:ipv4')
allowed = await prox.allowed(perm)
if allowed:
    dostuff()
```

Returns
True if the user has permission, False if explicitly denied, None if no entry

Return type
Optional[bool]

behold()

Yield Cell system messages

checkUserApiKey(*key*)

cullNexsLog(*offs*)

Remove Nexus log entries up to (and including) the given offset.

Note: If there are consumers of this cell's nexus log they must be caught up to at least the offs argument before culling.

Only rotated logs where the last index is less than the provided offset will be removed from disk.

Parameters
offs (*int*) – The offset to remove entries up to.

Returns
Whether the cull was executed

Return type
bool

delAuthRole(*name*)

`delAuthRule`(*name*, *rule*, *gateiden=None*)
This API is deprecated.

`delAuthUser`(*name*)

`delBackup`(*name*)
Delete a backup by name.

Parameters
`name` (*str*) – The name of the backup to delete.

`delRole`(*iden*)

`delRoleRule`(*iden*, *rule*, *gateiden=None*)

`delUser`(*iden*)

`delUserRole`(*useriden*, *roleiden*)

`delUserRule`(*iden*, *rule*, *gateiden=None*)

`dyncall`(*iden*, *todo*, *gatekeys=()*)

`dyniter`(*iden*, *todo*, *gatekeys=()*)

`genUserOnepass`(*iden*, *duration=60000*)

`getAuthGate`(*iden*)

`getAuthGates`()

`getAuthInfo`(*name*)
This API is deprecated.

`getAuthRoles`()

`getAuthUsers`(*archived=False*)

Parameters
`archived` (*bool*) – If true, list all users, else list non-archived users

`getBackupInfo`()
Get information about recent backup activity.

Returns

- currduration - If backup currently running, time in ms since backup started, otherwise None
- laststart - Last time (in epoch milliseconds) a backup started
- lastend - Last time (in epoch milliseconds) a backup ended
- lastduration - How long last backup took in ms
- lastsize - Disk usage of last backup completed
- lastupload - Time a backup was last completed being uploaded via iter(New)BackupArchive
- lastexception - Tuple of exception information if last backup failed, otherwise None

Return type
(dict) It has the following keys

Note: these statistics are not persistent, i.e. they are not preserved between cell restarts.

getBackups()

Retrieve a list of backups.

Returns
A list of backup names.

Return type
list[str]

getCellIden()

async **getCellInfo**()

async **getCellRunId**()

getCellType()

getCellUser()

getDiagInfo()

getDmonSessions()

getGcInfo()

For diagnostic purposes only!

NOTE: This API is *not* supported and can be removed at any time!

async **getHealthCheck**()

getHiveKey(*path*)

getHiveKeys(*path*)

getMirrorUrls()

getNexsIndx()

getNexusChanges(*offs*, *tellready=False*)

getPermDef(*perm*)

Return a specific permission definition.

getPermDefs()

Return a non-comprehensive list of perm definitions.

getReloadableSystems()

getRoleDef(*iden*)

getRoleDefByName(*name*)

getRoleDefs()

async **getRoleInfo**(*name*)

getSystemInfo()

Get info about the system in which the cell is running

Returns

- volsize - Volume where cell is running total space
- volfree - Volume where cell is running free space
- backupvolsize - Backup directory volume total space
- backupvolfree - Backup directory volume free space
- celluptime - Cell uptime in milliseconds
- cellrealdisk - Cell's use of disk, equivalent to du
- cellapprdisk - Cell's apparent use of disk, equivalent to ls -l
- osversion - OS version/architecture
- pyversion - Python version
- totalmem - Total memory in the system
- availmem - Available memory in the system

Return type

A dictionary with the following keys. All size values are in bytes

getUserDef(*iden*, *packroles=True*)

getUserDefByName(*name*)

getUserDefs()

async **getUserInfo**(*name*)

getUserProfInfo(*iden*, *name*)

getUserProfile(*iden*)

handoff(*turl*, *timeout=30*)

async **initCellApi**()

async **isCellActive**()

Returns True if the cell is an active/leader cell.

isRoleAllowed(*iden*, *perm*, *gateiden=None*)

isUserAllowed(*iden*, *perm*, *gateiden=None*, *default=False*)

issue(*nexsiden: str*, *event: str*, *args*, *kwargs*, *meta=None*)

iterBackupArchive(*name*)

Retrieve a backup by name as a compressed stream of bytes.

Note: Compression and streaming will occur from a separate process.

Parameters

name (*str*) – The name of the backup to retrieve.

`iterNewBackupArchive`(*name=None*, *remove=False*)

Run a new backup and return it as a compressed stream of bytes.

Note: Compression and streaming will occur from a separate process.

Parameters

- **name** (*str*) – The name of the backup to retrieve.
- **remove** (*bool*) – Delete the backup after streaming.

`async kill`(*iden*)

`listHiveKey`(*path=None*)

`popHiveKey`(*path*)

`popUserProfInfo`(*iden*, *name*, *default=None*)

`promote`(*graceful=False*)

`async ps`()

`readyToMirror`()

`reload`(*subsystem=None*)

`rotateNexsLog`()

Rotate the Nexus log at the current offset.

Returns
The starting index of the active Nexus log

Return type
int

`runBackup`(*name=None*, *wait=True*)

Run a new backup.

Parameters

- **name** (*str*) – The optional name of the backup.
- **wait** (*bool*) – On True, wait for backup to complete before returning.

Returns
The name of the newly created backup.

Return type
str

`runGcCollect`(*generation=2*)

For diagnostic purposes only!

NOTE: This API is *not* supported and can be removed at any time!

`saveHiveTree`(*path=()*)

`setAuthAdmin`(*name*, *isadmin*)

This API is deprecated.

setCellUser(*iden*)

Switch to another user (admin only).

This API allows remote admin/service accounts to impersonate a user. Used mostly by services that manage their own authentication/sessions.

setHiveKey(*path*, *valu*)

setRoleRules(*iden*, *rules*, *gateiden=None*)

setUserAdmin(*iden*, *admin*, *gateiden=None*)

setUserArchived(*useriden*, *archived*)

setUserEmail(*useriden*, *email*)

setUserLocked(*useriden*, *locked*)

async **setUserPasswd**(*iden*, *passwd*)

setUserProfInfo(*iden*, *name*, *valu*)

setUserRoles(*useriden*, *roleidens*)

setUserRules(*iden*, *rules*, *gateiden=None*)

trimNexsLog(*consumers=None*, *timeout=60*)

Rotate and cull the Nexus log (and those of any consumers) at the current offset.

Note: If the consumers argument is provided they will first be checked if online before rotating and raise otherwise. After rotation, all consumers must catch-up to the offset to cull at before executing the cull, and will raise otherwise.

Parameters

- **consumers** (*list or None*) – Optional list of telepath URLs for downstream Nexus log consumers.
- **timeout** (*int*) – Time in seconds to wait for downstream consumers to be caught up.

Returns

The offset that the Nexus log was culled up to and including.

Return type

int

tryUserPasswd(*name*, *passwd*)

waitNexsOffs(*offs*, *timeout=None*)

Wait for the Nexus log to write an offset.

Parameters

- **offs** (*int*) – The offset to wait for.
- **timeout** (*int or None*) – An optional timeout in seconds.

Returns

True if the offset was written, False if it timed out.

Return type
bool

synapse.lib.cell.**SSLCTX_CACHE_SIZE = 64**

Base classes for the synapse "cell" microservice architecture.

synapse.lib.cell.**adminapi**(*log=False*)

Decorator for CellApi (and subclasses) for requiring a method to be called only by an admin user.

Parameters
log (*bool*) – If set to True, log the user, function and arguments.

synapse.lib.certdir module

class synapse.lib.certdir.**CertDir**(*path: str | None = None*)

Bases: object

Certificate loading/generation/signing utilities.

Features:

- Locates and load certificates, keys, and certificate signing requests (CSRs).
- Generates keypairs for users, hosts, and certificate authorities (CAs), supports both signed and self-signed.
- Generates certificate signing requests (CSRs) for users, hosts, and certificate authorities (CAs).
- Signs certificate signing requests (CSRs).
- Generates PKCS#12 archives for use in browser.

Parameters
path (*str*) – Optional path which can override the default path directory.

Notes

- All certificates will be loaded from and written to ~/.syn/certs by default. Set the environment variable SYN_CERT_DIR to override.
- All certificate generation methods create 4096 bit RSA keypairs.
- All certificate signing methods use sha256 as the signature algorithm.
- CertDir does not currently support signing CA CSRs.

addCertPath(**path: str*)

delCertPath(**path: str*)

genCaCert(*name: str, signas: str | None = None, outp: OutPut | None = None, save: bool = True*) → Tuple[RSAPrivateKey, Certificate]

Generates a CA keypair.

Parameters

- **name** – The name of the CA keypair.
- **signas** – The CA keypair to sign the new CA with.

- **outp** – The output buffer.
- **save** – Save the certificate and key to disk.

Examples

Make a CA named "myca":

```
mycakey, mycacert = cdir.genCaCert('myca')
```

Returns
Tuple containing the private key and certificate objects.

genCaCrl(*name: str*) → *Crl*

Get the CRL for a given CA.

Parameters
name – The CA name.

Returns
The CRL object.

genClientCert(*name: str*, *outp:* OutPut | *None* = *None*) → None

Generates a user PKCS #12 archive.

Please note that the resulting file will contain private key material.

Parameters

- **name** (*str*) – The name of the user keypair.
- **outp** (*synapse.lib.output.Output*) – The output buffer.

Examples

Make the PKC12 object for user "myuser":

```
myuserpkcs12 = cdir.genClientCert('myuser')
```

Returns
None

genCodeCert(*name: str*, *signas: str* | *None* = *None*, *outp:* OutPut | *None* = *None*, *save: bool* = *True*) → Tuple[RSAPrivateKey, Certificate]

Generates a code signing keypair.

Parameters

- **name** – The name of the code signing cert.
- **signas** – The CA keypair to sign the new code keypair with.
- **outp** – The output buffer.

Examples

Generate a code signing cert for the name "The Vertex Project":

```
myuserkey, myusercert = cdir.genCodeCert('The Vertex Project')
```

Returns
Tuple containing the key and certificate objects.

genCrlPath(*name: str*) → str

genHostCert(*name: str*, *signas: str | None = None*, *outp: OutPut | None = None*, *csr: DHPublicKey | DSAPublicKey | RSAPublicKey | EllipticCurvePublicKey | Ed25519PublicKey | Ed448PublicKey | X25519PublicKey | X448PublicKey | None = None*, *sans: str | None = None*, *save: bool = True*) → Tuple[RSAPrivateKey | None, Certificate]

Generates a host keypair.

Parameters

- **name** – The name of the host keypair.
- **signas** – The CA keypair to sign the new host keypair with.
- **outp** – The output buffer.
- **csr** – The CSR public key when generating the keypair from a CSR.
- **sans** – String of comma separated alternative names.

Examples

Make a host keypair named "myhost":

```
myhostkey, myhostcert = cdir.genHostCert('myhost')
```

Returns
Tuple containing the private key and certificate objects. Private key may be None when signing a CSR.

genHostCsr(*name: str*, *outp: OutPut | None = None*) → bytes

Generates a host certificate signing request.

Parameters

- **name** – The name of the host CSR.
- **outp** – The output buffer.

Examples

Generate a CSR for the host key named "myhost":

```
cdir.genHostCsr('myhost')
```

Returns
The bytes of the CSR.

genUserCert(*name: str*, *signas: str | None = None*, *outp: OutPut | None = None*, *csr: DHPublicKey | DSAPublicKey | RSAPublicKey | EllipticCurvePublicKey | Ed25519PublicKey | Ed448PublicKey | X25519PublicKey | X448PublicKey | None = None*, *save: bool = True*) → Tuple[RSAPrivateKey | None, Certificate]

Generates a user keypair.

Parameters

- **name** – The name of the user keypair.
- **signas** – The CA keypair to sign the new user keypair with.
- **outp** – The output buffer.
- **csr** – The CSR public key when generating the keypair from a CSR.

Examples

Generate a user cert for the user "myuser":

```
myuserkey, myusercert = cdir.genUserCert('myuser')
```

Returns
Tuple containing the key and certificate objects.

genUserCsr(*name: str*, *outp: OutPut | None = None*) → bytes

Generates a user certificate signing request.

Parameters

- **name** – The name of the user CSR.
- **outp** – The output buffer.

Examples

Generate a CSR for the user "myuser":

```
cdir.genUserCsr('myuser')
```

Returns
The bytes of the CSR.

getCaCert(*name: str*) → Certificate | None

Loads the X509 object for a given CA.

Parameters
name – The name of the CA keypair.

Examples

Get the certificate for the CA "myca":

```
mycacert = cdir.getCaCert('myca')
```

Returns
The certificate, if exists.

getCaCertBytes(*name: str*) → bytes

getCaCertPath(*name: str*) → str | None

Gets the path to a CA certificate.

Parameters
name – The name of the CA keypair.

Examples

Get the path to the CA certificate for the CA "myca":

```
mypath = cdir.getCACertPath('myca')
```

Returns
The path, if exists.

getCaCerts() → List[Certificate]

Return a list of CA certs from the CertDir.

Returns
List of CA certificates.

getCaKey(*name*) → RSAPrivateKey | DSAPrivateKey | None

Loads the PKey object for a given CA keypair.

Parameters
name – The name of the CA keypair.

Examples

Get the private key for the CA "myca":

```
mycakey = cdir.getCaKey('myca')
```

Returns
The private key, if exists.

getCaKeyPath(*name: str*) → str | None

Gets the path to a CA key.

Parameters
name – The name of the CA keypair.

Examples

Get the path to the private key for the CA "myca":

```
mypath = cdir.getCAKeyPath('myca')
```

Returns
The path, if exists.

getClientCert(*name: str*) → PKCS12KeyAndCertificates | None

Loads the PKCS12 archive object for a given user keypair.

Parameters
name – The name of the user keypair.

Examples

Get the PKCS12 object for the user "myuser":

```
mypkcs12 = cdir.getClientCert('myuser')
```

Notes

The PKCS12 archive will contain private key material if it was created with CertDir or the easycert tool

Returns
The PKCS12 archive, if exists.

getClientCertPath(*name: str*) → str | None

Gets the path to a client certificate.

Parameters
name – The name of the client keypair.

Examples

Get the path to the client certificate for "myuser":

```
mypath = cdir.getClientCertPath('myuser')
```

Returns
The path, if exists.

getClientSSLContext(*certname: str | None = None*) → SSLContext

Returns an ssl.SSLContext appropriate for initiating a TLS session

Parameters
certname – If specified, use the user certificate with the matching name to authenticate to the remote service.

Returns
A SSLContext object.

getCodeCert(*name: str*) → Certificate | None

getCodeCertPath(*name: str*) → str | None

getCodeKey(*name: str*) → *PriKey* | None

getCodeKeyPath(*name: str*) → str | None

getCrlPath(*name: str*) → str | None

getHostCaPath(*name: str*) → str | None

Gets the path to the CA certificate that issued a given host keypair.

Parameters
name – The name of the host keypair.

Examples

Get the path to the CA cert which issue the cert for "myhost":

```
mypath = cdir.getHostCaPath('myhost')
```

Returns
The path, if exists.

getHostCert(*name: str*) → Certificate | None

Loads the X509 object for a given host keypair.

Parameters
name – The name of the host keypair.

Examples

Get the certificate object for the host "myhost":

```
myhostcert = cdir.getHostCert('myhost')
```

Returns
The certificate, if exists.

getHostCertHash(*name: str*) → str | None

getHostCertPath(*name: str*) → str | None

Gets the path to a host certificate.

Parameters
name – The name of the host keypair.

Examples

Get the path to the host certificate for the host "myhost":

```
mypath = cdir.getHostCertPath('myhost')
```

Returns
The path, if exists.

getHostCsrPath(*name: str*) → str | None

getHostKey(*name: str*) → RSAPrivateKey | DSAPrivateKey | None

Loads the PKey object for a given host keypair.

Parameters
name – The name of the host keypair.

Examples

Get the private key object for the host "myhost":

```
myhostkey = cdir.getHostKey('myhost')
```

Returns
The private key, if exists.

getHostKeyPath(*name: str*) → str | None

Gets the path to a host key.

Parameters
name – The name of the host keypair.

Examples

Get the path to the host key for the host "myhost":

```
mypath = cdir.getHostKeyPath('myhost')
```

Returns
The path if exists.

Return type
str

getServerSSLContext(*hostname: str | None = None*, *caname: str | None = None*) → SSLContext

Returns an ssl.SSLContext appropriate to listen on a socket

Parameters

- **hostname** – If None, the value from socket.gethostname is used to find the key in the servers directory. This name should match the not-suffixed part of two files ending in .key and .crt in the hosts subdirectory.
- **caname** – If not None, the given name is used to locate a CA certificate used to validate client SSL certs.

Returns
A SSLContext object.

getUserCaPath(*name: str*) → str | None

Gets the path to the CA certificate that issued a given user keypair.

Parameters
name – The name of the user keypair.

Examples

Get the path to the CA cert which issue the cert for "myuser":

```
mypath = cdir.getUserCaPath('myuser')
```

Returns
The path, if exists.

getUserCert(*name: str*) → Certificate | None

Loads the X509 object for a given user keypair.

Parameters
name – The name of the user keypair.

Examples

Get the certificate object for the user "myuser":

```
myusercert = cdir.getUserCert('myuser')
```

Returns
The certificate, if exists.

getUserCertPath(*name: str*) → str | None

Gets the path to a user certificate.

Parameters
name (*str*) – The name of the user keypair.

Examples

Get the path for the user cert for "myuser":

```
mypath = cdir.getUserCertPath('myuser')
```

Returns
The path, if exists.

getUserCsrPath(*name: str*) → str | None

getUserForHost(*user: str*, *host: str*) → str | None

Gets the name of the first existing user cert for a given user and host.

Parameters

- **user** – The name of the user.
- **host** – The name of the host.

Examples

Get the name for the "myuser" user cert at "cool.vertex.link":

```
usercertname = cdir.getUserForHost('myuser', 'cool.vertex.link')
```

Returns
The cert name, if exists.

Return type
str

getUserKey(*name: str*) → RSAPrivateKey | DSAPrivateKey | None

Loads the PKey object for a given user keypair.

Parameters
name – The name of the user keypair.

Examples

Get the key object for the user key for "myuser":

```
myuserkey = cdir.getUserKey('myuser')
```

Returns
The private key, if exists.

getUserKeyPath(*name: str*) → str | None

Gets the path to a user key.

Parameters
name – The name of the user keypair.

Examples

Get the path to the user key for "myuser":

```
mypath = cdir.getUserKeyPath('myuser')
```

Returns
The path, if exists.

importFile(*path: str*, *mode: str*, *outp: OutPut | None = None*) → None

Imports certs and keys into the Synapse cert directory

Parameters

- **path** – The path of the file to be imported.
- **mode** – The certdir subdirectory to import the file into.

Examples

Import CA certifciate 'mycoolca.crt' to the 'cas' directory:

```
certdir.importFile('mycoolca.crt', 'cas')
```

Notes

importFile does not perform any validation on the files it imports.

Returns
None

isCaCert(*name: str*) → bool

Checks if a CA certificate exists.

Parameters
name – The name of the CA keypair.

Examples

Check if the CA certificate for "myca" exists:

```
exists = cdir.isCaCert('myca')
```

Returns
True if the certificate is present, False otherwise.

isClientCert(*name: str*) → bool

Checks if a user client certificate (PKCS12) exists.

Parameters
name – The name of the user keypair.

Examples

Check if the client certificate "myuser" exists:

```
exists = cdir.isClientCert('myuser')
```

Returns
True if the certificate is present, False otherwise.

isCodeCert(*name: str*) → bool

Checks if a code certificate exists.

Parameters
name – The name of the code keypair.

Examples

Check if the code cert "mypipeline" exists:

```
exists = cdir.isCodeCert('mypipeline')
```

Returns
True if the certificate is present, False otherwise.

isHostCert(*name: str*) → bool

Checks if a host certificate exists.

Parameters
name – The name of the host keypair.

Examples

Check if the host cert "myhost" exists:

```
exists = cdir.isUserCert('myhost')
```

Returns
True if the certificate is present, False otherwise.

isUserCert(*name: str*) → bool

Checks if a user certificate exists.

Parameters
name – The name of the user keypair.

Examples

Check if the user cert "myuser" exists:

```
exists = cdir.isUserCert('myuser')
```

Returns
True if the certificate is present, False otherwise.

loadCertByts(*byts: bytes*) → Certificate

Load a X509 certificate from its PEM encoded bytes.

Parameters
byts – The PEM encoded bytes of the certificate.

Returns
The X509 certificate.

Raises
BadCertBytes – If the certificate bytes are invalid.

saveCaCertByts(*byts: bytes*) → str

saveCertPem(*cert: Certificate*, *path: str*) → None

Save a certificate in PEM format to a file outside the certdir.

saveCodeCertBytes(*byts: bytes*) → str

saveHostCertByts(*byts: bytes*) → str

savePkeyPem(*pkey: DHPrivateKey | Ed25519PrivateKey | Ed448PrivateKey | RSAPrivateKey | DSAPrivateKey | EllipticCurvePrivateKey | X25519PrivateKey | X448PrivateKey*, *path: str*) → None

Save a private key in PEM format to a file outside the certdir.

saveUserCertByts(*byts: bytes*) → str

selfSignCert(*builder: CertificateBuilder*, *pkey: RSAPrivateKey | DSAPrivateKey*) → Certificate

Self-sign a certificate.

Parameters

- **cert** – The certificate to sign.
- **pkey** – The PKey with which to sign the certificate.

Examples

Sign a given certificate with a given private key:

```
cdir.selfSignCert(mycert, myotherprivatekey)
```

Returns
None

signCertAs(*builder: CertificateBuilder*, *signas: str*) → Certificate

Signs a certificate with a CA keypair.

Parameters

- **cert** – The certificate to sign.
- **signas** – The CA keypair name to sign the new keypair with.

Examples

Sign a certificate with the CA "myca":

```
cdir.signCertAs(mycert, 'myca')
```

Returns
None

signHostCsr(*xcsr: CertificateSigningRequest*, *signas: str*, *outp: OutPut | None = None*, *sans: str | None = None*, *save: bool = True*) → Tuple[RSAPrivateKey | None, Certificate]

Signs a host CSR with a CA keypair.

Parameters

- **xcsr** – The certificate signing request.
- **signas** – The CA keypair name to sign the CSR with.
- **outp** – The output buffer.
- **sans** – List of subject alternative names.

Examples

Sign a host key with the CA "myca":

```
cdir.signHostCsr(mycsr, 'myca')
```

Returns
Tuple containing the public key and certificate objects.

signUserCsr(*xcsr: CertificateSigningRequest*, *signas: str*, *outp: OutPut | None = None*, *save: bool = True*) → Tuple[RSAPrivateKey | None, Certificate]

Signs a user CSR with a CA keypair.

Parameters

- **xcsr** – The certificate signing request.
- **signas** – The CA keypair name to sign the CSR with.
- **outp** – The output buffer.

Examples

Sign a user CSR with "myca":

```
cdir.signUserCsr(mycsr, 'myca')
```

Returns
Tuple containing the public key and certificate objects.

Return type
((OpenSSL.crypto.PKey, OpenSSL.crypto.X509))

valCodeCert(*byts: bytes*) → Certificate

Verify a code cert is valid according to certdir's available CAs and CRLs.

Parameters
byts – The certificate bytes.

Raises
s_exc.BadCertVerify if we are unable to verify the certificate. –

Returns
The certificate.

valUserCert(*byts: bytes*, *cacerts: List[Certificate] | None = None*) → Certificate

Validate the PEM encoded x509 user certificate bytes and return it.

Parameters

- **byts** – The bytes for the User Certificate.
- **cacerts** – A tuple of CA Certificates to use for validating the user cert.

Raises
BadCertVerify – If the certificate is not valid.

Returns
The certificate, if it is valid.

class synapse.lib.certdir.**Crl**(*cdir*, *name*)

Bases: object

revoke(*cert: Certificate*) → None

Revoke a certificate with the CRL.

Parameters
cert – The certificate to revoke.

Returns
None

synapse.lib.certdir.**addCertPath**(*path*)

synapse.lib.certdir.**delCertPath**(*path*)

synapse.lib.certdir.**getCertDir**() → *CertDir*

Get the singleton CertDir instance.

Returns
A certdir object.

Return type
CertDir

synapse.lib.certdir.**getCertDirn**() → str

Get the expanded default path used by the singleton CertDir instance.

Returns
The path string.

Return type
str

synapse.lib.certdir.**getServerSSLContext**() → SSLContext

Get a server SSLContext object.

This object has a minimum TLS version of 1.2, a subset of ciphers in use, and disabled client renegotiation.

This object has no certificates loaded in it.

Returns
The context object.

Return type
ssl.SSLContext

synapse.lib.certdir.**iterFqdnUp**(*fqdn*)

synapse.lib.chop module

synapse.lib.chop.**TagMatchRe** = **regex.Regex('([\\w*]+\\.)*[\\w*]+', flags=regex.V0)**

Shared primitive routines for chopping up strings and values.

synapse.lib.chop.**cvss2_normalize**(*vect*)

Helper function to normalize CVSS2 vectors

synapse.lib.chop.**cvss3x_normalize**(*vect*)

Helper function to normalize CVSS3.X vectors

synapse.lib.chop.**cvss_normalize**(*vdict*, *vers*)

Normalize CVSS vectors

synapse.lib.chop.**cvss_validate**(*vect*, *vers*)

Validate (as best as possible) the CVSS vector string. Look for issues such as:

- No duplicated metrics
- Invalid metrics

- Invalid metric values
- Missing mandatory metrics

Returns a dictionary with the parsed metric:value pairs.

synapse.lib.chop.**digits**(*text*)

synapse.lib.chop.**hexstr**(*text*)

Ensure a string is valid hex.

Parameters
text (*str*) – String to normalize.

Examples

Norm a few strings:

hexstr('0xff00') hexstr('ff00')

Notes

Will accept strings prefixed by '0x' or '0X' and remove them.

Returns
Normalized hex string.

Return type
str

synapse.lib.chop.**intstr**(*text*)

synapse.lib.chop.**onespace**(*text*)

synapse.lib.chop.**printables**(*text*)

synapse.lib.chop.**replaceUnicodeDashes**(*valu*)

Replace unicode dashes in a string with regular dashes.

Parameters
valu (*str*) – A string.

Returns
A new string with replaced dashes.

Return type
str

synapse.lib.chop.**stormstring**(*s*)

Make a string storm safe by escaping backslashes and double quotes.

Parameters
s (*str*) – String to make storm safe.

Notes

This does not encapsulate a string in double quotes.

Returns
A string which can be embedded directly into a storm query.

Return type
str

`synapse.lib.chop.tag(text)`

`synapse.lib.chop.tagpath(text)`

`synapse.lib.chop.tags(norm)`

Divide a normalized tag string into hierarchical layers.

`synapse.lib.chop.uncnorm(valu)`

Validate and normalize the UNC path passed in *valu* into a URI.

This function will accept *@SSL* and *@<port>* as part of the host name to indicate SSL (https) or a specific port number. It can also accept IPv6 addresses in the host name even though those are non-standard according to the spec.

`synapse.lib.chop.validateTagMatch(tag)`

Raises an exception if tag is not a valid tagmatch (i.e. a tag that might have globs)

synapse.lib.cli module

`class synapse.lib.cli.Cli`

Bases: `Base`

A modular / event-driven CLI base object.

`addCmdClass(ctor, **opts)`

Add a Cmd subclass to this cli.

`async addSignalHandlers()`

Register SIGINT signal handler with the ioloop to cancel the currently running cmdloop task.

`get(name, defval=None)`

`getCmdByName(name)`

Return a Cmd instance by name.

`getCmdNames()`

Return a list of all the known command names for the CLI.

`getCmdPrompt()`

Get the command prompt.

Returns
Configured command prompt

Return type
str

`histfile = 'cmdr_history'`

initCmdClasses()

printf(*mesg*, *addnl=True*, *color=None*)

async **prompt**(*text=None*)

Prompt for user input from stdin.

async **runCmdLine**(*line*)

Run a single command line.

Parameters

`line` (*str*) – Line to execute.

Examples

Execute the 'woot' command with the 'help' switch:

await cli.runCmdLine('woot –help')

Returns

Arbitrary data from the cmd class.

Return type

object

async **runCmdLoop**()

Run commands from a user in an interactive fashion until fini() or EOFError is raised.

set(*name*, *valu*)

class synapse.lib.cli.**Cmd**(*cli*, ***opts*)

Bases: `object`

Base class for modular commands in the synapse CLI.

getCmdBrief()

Return the single-line description for this command.

getCmdDoc()

Return the help/doc output for this command.

getCmdItem()

Get a reference to the object we are commanding.

getCmdName()

getCmdOpts(*text*)

Use the _cmd_syntax def to split/parse/normalize the cmd line.

Parameters

`text` (*str*) – Command to process.

Notes

This is implemented independent of argparse (et al) due to the need for syntax aware argument splitting. Also, allows different split per command type

Returns
An opts dictionary.

Return type
dict

`printf`(*mesg*, *addnl=True*, *color=None*)

async `runCmdLine`(*line*)

Run a line of command input for this command.

Parameters
line (*str*) – Line to execute

Examples

Run the foo command with some arguments:

await foo.runCmdLine('foo –opt baz woot.com')

async `runCmdOpts`(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.lib.cli.`CmdHelp`(*cli*, ***opts*)

Bases: *Cmd*

List commands and display help output.

Example

help foocmd

async `runCmdOpts`(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.lib.cli.`CmdLocals`(*cli*, ***opts*)

Bases: *Cmd*

List the current locals for a given CLI object.

async `runCmdOpts`(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.lib.cli.**CmdQuit**(*cli*, ***opts*)

Bases: *Cmd*

Quit the current command line interpreter.

Example

quit

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters

opts (*dict*) – Options dictionary.

synapse.lib.cmd module

class synapse.lib.cmd.**Parser**(*prog=None*, *outp=<synapse.lib.output.OutPut object>*, ***kwargs*)

Bases: ArgumentParser

exit(*status=0*, *message=None*)

Argparse expects exit() to be a terminal function and not return. As such, this function must raise an exception instead.

synapse.lib.cmdr module

async synapse.lib.cmdr.**getItemCmdr**(*cell*, *outp=None*, *color=False*, ***opts*)

Construct and return a cmdr for the given remote cell.

Parameters

- **cell** – Cell proxy being commanded.
- **outp** – Output helper object.
- **color** (*bool*) – If true, enable colorized output.
- ****opts** – Additional options pushed into the Cmdr locs.

Examples

Get the cmdr for a proxy:

```
cmdr = await getItemCmdr(foo)
```

Returns

A Cli instance with Cmds loaeded into it.

Return type

s_cli.Cli

async synapse.lib.cmdr.**runItemCmdr**(*item*, *outp=None*, *color=False*, ***opts*)

Create a cmdr for the given item and run the cmd loop.

Parameters

- **item** – Cell proxy being commanded.
- **outp** – Output helper object.
- **color** (*bool*) – If true, enable colorized output.
- ****opts** – Additional options pushed into the Cmdr locs.

Notes

This function does not return while the command loop is run.

Examples

Run the Cmdr for a proxy:

```
await runItemCmdr(foo)
```

Returns
This function returns None.

Return type
None

synapse.lib.config module

class synapse.lib.config.**Config**(*schema*, *conf=None*, *envar_prefixes=None*)

Bases: `MutableMapping`

Synapse configuration helper based on JSON Schema.

Parameters

- **schema** (*dict*) – The JSON Schema (draft v7) which to validate configuration data against.
- **conf** (*dict*) – Optional, a set of configuration data to preload.
- **envar_prefixes** (*list*) – Optional, a list of prefix strings used when collecting configuration data from environment variables.

Notes

This class implements the collections.abc.MutableMapping class, so it may be used where a dictionary would otherwise be used.

The default values provided in the schema must be able to be recreated from the repr() of their Python value.

Default values are not loaded into the configuration data until the `reqConfValid()` method is called.

asDict()

Get a copy of configuration data.

Returns
A copy of the configuration data.

Return type
dict

getArgParseArgs()

getCmdlineMapping()

classmethod getConfFromCell(*cell*, *conf=None*, *envar_prefixes=None*)

Get a Config object from a Cell directly (either the ctor or the instance thereof).

Returns
A Config object.

Return type
Config

getEnvarMapping(*prefix=None*)

Get a mapping of config values to envars.

Configuration values which have the `hideconf` value set to True are not resolved from environment variables.

reqConfValid()

Validate that the loaded configuration data is valid according to the schema.

Notes

The validation set does set any default values which are not currently set for configuration options.

Returns
This returns nothing.

Return type
None

reqConfValu(*key*)

Get a configuration value. If that value is not present in the schema or is not set, then raise an exception.

Parameters
key (*str*) – The key to require.

Returns
The requested value.

reqKeyValid(*key*, *value*)

Test if a key is valid for the provided schema it is associated with.

Parameters

- **key** (*str*) – Key to check.
- **value** – Value to check.

Raises

- *BadArg* – If the key has no associated schema.

- *BadConfValu* – If the data is not schema valid.

Returns
None when valid.

`setConfFromEnvs()`

Set configuration options from environment variables.

Notes

Environment variables are resolved from configuration options after doing the following transform:

- Replace `:` characters with `_`.
- Add a config provided prefix, if set.
- Uppercase the string.
- Resolve the environment variable
- If the environment variable is set, set the config value to the results of `yaml.yaml_safeload()` on the value.

Configuration values which have the `hideconf` value set to True are not resolved from environment variables.

Examples

For the configuration value `auth:passwd`, the environment variable is resolved as `AUTH_PASSWD`. With the prefix `cortex`, the the environment variable is resolved as `CORTEX_AUTH_PASSWD`.

Returns
Returns a dictionary of values which were set from enviroment variables.

Return type
dict

`setConfFromFile(path, force=False)`

Set the opts for a conf object from YAML file path.

Parameters

- **path** (*str*) – Path to the yaml load. If it exists, it must represent a dictionary.
- **force** (*bool*) – Force the update instead of using setdefault() behavior.

Returns
None

`setConfFromOpts(opts=None)`

Set the opts for a conf object from a namespace object.

Parameters

- **opts** (*argparse.Namespace*) – A Namespace object made from parsing args with an ArgumentParser
- **getArgumentParser.** (*made with*) –

Returns
Returns None.

Return type
None

`synapse.lib.config.getJsSchema(confbase, confdefs)`

Generate a Synapse JSON Schema for a Cell using a pair of confbase and confdef values.

Parameters

- **confbase** (*dict*) – A JSON Schema dictionary of properties for the object. This content has precedence over the confdefs argument.
- **confdefs** (*dict*) – A JSON Schema dictionary of properties for the object.

Notes

This generated a JSON Schema draft 7 schema for a single object, which does not allow for additional properties to be set on it. The data in confdefs is implementer controlled and is welcome to specify

Returns
A complete JSON schema.

Return type
dict

`synapse.lib.config.getJsValidator(schema, use_default=True, handlers={'http': <function localSchemaRefHandler>, 'https': <function localSchemaRefHandler>})`

Get a fastjsonschema callable.

Parameters

- **schema** (*dict*) – A JSON Schema object.
- **use_default** (*bool*) – Whether to insert "default" key arguments into the validated data structure.

Returns
A callable function that can be used to validate data against the json schema.

Return type
callable

`synapse.lib.config.localSchemaRefHandler(uri)`

This function parses the given URI to get the path component and then tries to resolve the referenced schema from the 'jsonschemas' directory of synapse.data.

`synapse.lib.config.make_envar_name(key, prefix=None)`

Convert a colon delimited string into an uppercase, underscore delimited string.

Parameters

- **key** (*str*) – Config key to convert.
- **prefix** (*str*) – Optional string prefix to prepend the the config key.

Returns
The string to lookup against a envar.

Return type
str

synapse.lib.const module

synapse.lib.coro module

Async/Coroutine related utilities.

class synapse.lib.coro.**Event**

Bases: Event

async timewait(*timeout=None*)

class synapse.lib.coro.**GenrHelp**(*genr*)

Bases: object

async list()

async spin()

async synapse.lib.coro.**agen**(*item*)

Wrap an async_generator *or* generator in an async_generator.

Notes

Do not use this for a synchronous generator which would cause non-blocking IO; otherwise that IO will block the ioloop.

async synapse.lib.coro.**event_wait**(*event: Event*, *timeout=None*)

Wait on an an asyncio event with an optional timeout

Returns

true if the event got set, False if timed out

synapse.lib.coro.**executor**(*func*, **args*, ***kwargs*)

Execute a non-coroutine function in the ioloop executor pool.

Parameters

- **func** – Function to execute.
- ***args** – Args for the function.
- ****kwargs** – Kwargs for the function.

Examples

Execute a blocking API call in the executor pool:

```
import requests

def block(url, params=None):
    return requests.get(url, params=params).json()

fut = s_coro.executor(block, 'http://some.tld/thign')
resp = await fut
```

Returns
An asyncio future.

Return type
asyncio.Future

async synapse.lib.coro.**forked**(*func*, **args*, ***kwargs*)

Execute a target function in the shared forked process pool and fallback to running in a spawned process if the pool is unavailable.

Parameters

- **func** – The target function.
- ***args** – Function positional arguments.
- ****kwargs** – Function keyword arguments.

Returns
The target function return.

synapse.lib.coro.**genrhelp**(*f*)

synapse.lib.coro.**iscoro**(*item*)

async synapse.lib.coro.**ornot**(*func*, **args*, ***kwargs*)

Calls func and awaits it if a returns a coroutine.

Note: This is useful for implementing a function that might take a telepath proxy object or a local object, and you must call a non-async method on that object.

This is also useful when calling a callback that might either be a coroutine function or a regular function.

Usage:
ok = await s_coro.ornot(maybeproxy.allowed, 'path')

async synapse.lib.coro.**semafork**(*func*, **args*, ***kwargs*)

Execute a target function in the shared forked process pool gated by a semaphore to ensure there are workers reserved for the Storm parser.

Parameters

- **func** – The target function.
- ***args** – Function positional arguments.
- ****kwargs** – Function keyword arguments.

Returns
The target function return.

synapse.lib.coro.**set_pool_logging**(*logger_*, *logconf*)

async synapse.lib.coro.**spawn**(*todo*, *timeout=None*, *ctx=None*, *log_conf=None*)

Run a todo (func, args, kwargs) tuple in a multiprocessing subprocess.

Parameters

- **todo** (*tuple*) – A tuple of function, `*args`, and `**kwargs`.
- **timeout** (*int*) – The timeout to wait for the todo function to finish.

- **ctx** (*multiprocess.Context*) – A optional multiprocessing context object.
- **log_conf** (*dict*) – An optional logging configuration for the spawned process.

Notes

The contents of the todo tuple must be able to be pickled for execution. This means that locally bound functions are not eligible targets for spawn.

Returns
The return value of executing the todo function.

async synapse.lib.coro.**waittask**(*task*, *timeout=None*)
Await a task without cancelling it when you time out.

Returns
True if the task completed before the timeout.

Return type
boolean

synapse.lib.datfile module

Utilities for handling data files embedded within python packages.

synapse.lib.datfile.**openDatFile**(*datpath*)
Open a file-like object using a pkg relative path.

Example

fd = openDatFile('foopkg.barpkg/wootwoot.bin')

synapse.lib.dyndeps module

synapse.lib.dyndeps.**getDynLocal**(*name*)
Dynamically import a python module and return a local.

Example

cls = getDynLocal('foopkg.barmod.BlahClass') blah = cls()

synapse.lib.dyndeps.**getDynMeth**(*name*)
Retrieve and return an unbound method by python path.

synapse.lib.dyndeps.**getDynMod**(*name*)
Dynamically import a python module and return a ref (or None).

Example

mod = getDynMod('foo.bar')

synapse.lib.dyndeps.**runDynTask**(*task*)

Run a dynamic task and return the result.

Example

foo = runDynTask(('baz.faz.Foo', (), { }))

synapse.lib.dyndeps.**tryDynFunc**(*name*, **args*, ***kwargs*)

Dynamically import a module and call a function or raise an exception.

synapse.lib.dyndeps.**tryDynLocal**(*name*)

Dynamically import a module and return a module local or raise an exception.

synapse.lib.dyndeps.**tryDynMod**(*name*)

Dynamically import a python module or exception.

synapse.lib.encoding module

synapse.lib.encoding.**addFormat**(*name*, *fn*, *opts*)

Add an additional ingest file format

synapse.lib.encoding.**decode**(*name*, *byts*, ***opts*)

Decode the given byts with the named decoder. If name is a comma separated list of decoders, loop through and do them all.

Example

byts = s_encoding.decode('base64',byts)

Note: Decoder names may also be prefixed with +
to *encode* for that name/layer.

synapse.lib.encoding.**encode**(*name*, *item*, ***opts*)

synapse.lib.encoding.**iterdata**(*fd*, *close_fd=True*, ***opts*)

Iterate through the data provided by a file like object.

Optional parameters may be used to control how the data is deserialized.

Examples

The following example show use of the iterdata function.:

```
with open('foo.csv','rb') as fd:
    for row in iterdata(fd, format='csv', encoding='utf8'):
        dostuff(row)
```

Parameters

- **fd** (*file*) – File like object to iterate over.

- **close_fd** (*bool*) – Default behavior is to close the fd object. If this is not true, the fd will not be closed.
- ****opts** (*dict*) – Ingest open directive. Causes the data in the fd to be parsed according to the 'format' key and any additional arguments.

Yields
An item to process. The type of the item is dependent on the format parameters.

synapse.lib.gis module

synapse.lib.gis.**bbox**(*lat*, *lon*, *dist*)

Calculate a min/max bounding box for the circle defined by lat/lon/dist.

Parameters

- **lat** (*float*) – The latitude in degrees
- **lon** (*float*) – The longitude in degrees
- **dist** (*int*) – A distance in geo:dist base units (mm)

Returns
(latmin, latmax, lonmin, lonmax)

Return type
(float,float,float,float)

synapse.lib.gis.**dms2dec**(*degs*, *mins*, *secs*)

Convert degrees, minutes, seconds lat/long form to degrees float.

Parameters

- **degs** (*int*) – Degrees
- **mins** (*int*) – Minutes
- **secs** (*int*) – Seconds

Returns
Degrees

Return type
(float)

synapse.lib.gis.**haversine**(*px*, *py*, *r=6371008800.0*)

Calculate the haversine distance between two points defined by (lat,lon) tuples.

Parameters

- **px** (*(float,float)*) – lat/long position 1
- **py** (*(float,float)*) – lat/long position 2
- **r** (*float*) – Radius of sphere

Returns
Distance in mm.

Return type
(int)

`synapse.lib.gis.`**`latlong`**(*text*)

Chop a latlong string and return (float,float). Does not perform validation on the coordinates.

Parameters

`text` (*`str`*) – A longitude,latitude string.

Returns

A longitude, latitude float tuple.

Return type

(float,float)

`synapse.lib.gis.`**`near`**(*point*, *dist*, *points*)

Determine if the given point is within dist of any of points.

Parameters

- **`point`** (*`(float,float)`*) – A latitude, longitude float tuple.
- **`dist`** (*`int`*) – A distance in mm (base units)
- **`points`** (*`list`*) – A list of latitude, longitude float tuples to compare against.

synapse.lib.grammar module

`synapse.lib.grammar.`**`chop_float`**(*text*, *off*)

`synapse.lib.grammar.`**`isBasePropNoPivprop`**(*name*)

`synapse.lib.grammar.`**`isCmdName`**(*name*)

`synapse.lib.grammar.`**`isFormName`**(*name*)

`synapse.lib.grammar.`**`isPropName`**(*name*)

`synapse.lib.grammar.`**`isUnivName`**(*name*)

`synapse.lib.grammar.`**`meh`**(*txt*, *off*, *cset*)

`synapse.lib.grammar.`**`nom`**(*txt*, *off*, *cset*, *trim=True*)

Consume chars in set from the string and return (subtxt,offset).

Example

text = "foo(bar)" chars = set('abcdefghijklmnopqrstuvwxyz')

name,off = nom(text,0,chars)

Note:

This really shouldn't be used for new code

`synapse.lib.grammar.`**`parse_float`**(*text*, *off*)

synapse.lib.hashitem module

synapse.lib.hashitem.**hashitem**(*item*)

Generate a uniq hash for the JSON compatible primitive data structure.

synapse.lib.hashitem.**normdict**(*item*)

synapse.lib.hashitem.**normitem**(*item*)

synapse.lib.hashitem.**normiter**(*item*)

synapse.lib.hashset module

class synapse.lib.hashset.**HashSet**

Bases: object

digests()

Get a list of (name, bytes) tuples for the hashes in the hashset.

eatfd(*fd*)

Consume all the bytes from a file like object.

Example

hset = HashSet() hset.eatfd(fd)

guid()

Use elements from this hash set to create a unique (re)identifier.

update(*byts*)

Update all the hashes in the set with the given bytes.

synapse.lib.health module

class synapse.lib.health.**HealthCheck**(*iden*)

Bases: object

getStatus()

pack()

setStatus(*valu*)

update(*name*, *status*, *mesg=''*, *data=None*)

Append a new component to the Healcheck object.

Parameters

- **name** (*str*) – Name of the reported component.
- **status** (*str*) – nomdinal/degraded/failed status code.
- **mesg** (*str*) – Optional message about the component status.
- **data** (*dict*) – Optional arbitrary dictionary of additional metadata about the component.

Returns
None

synapse.lib.hive module

`class synapse.lib.hive.Hive`

Bases: `Pusher`, `Aware`

An optionally persistent atomically accessed tree which implements primitives for use in making distributed/clustered services.

`async add(full, valu)`

Atomically increments a node's value.

`async dict(full, nexs=False)`

Open a HiveDict at the given full path.

Parameters
full (*tuple*) – A full path tuple.

Returns
A HiveDict for the full path.

Return type
HiveDict

`dir(full)`

List subnodes of the given Hive path.

Parameters
full (*tuple*) – A full path tuple.

Notes

This returns None if there is not a node at the path.

Returns
A list of tuples. Each tuple contains the name, node value, and the number of children nodes.

Return type
list

`async exists(full)`

Returns whether the Hive path has already been created.

`async get(full, defv=None)`

Get the value of a node at a given path.

Parameters
full (*tuple*) – A full path tuple.

Returns
Arbitrary node value.

async **getHiveAuth**()

Retrieve a HiveAuth for hive standalone or non-cell uses.

Note: This is for the hive's own auth, or for non-cell auth. It isn't the same auth as for a cell

async **getTeleApi**(*link*, *mesg*, *path*)

Return a shared object for this link. :param link: A network link. :type link: synapse.lib.link.Link :param mesg: The tele:syn handshake message. :type mesg: (str,dict)

async **loadHiveTree**(*tree*, *path=()*, *trim=False*)

async **open**(*full*)

Open and return a hive Node().

Parameters
full (*tuple*) – A full path tuple.

Returns
A Hive node.

Return type
Node

async **pop**(*full*, *nexs=False*)

Remove and return the value for the given node.

async **rename**(*oldpath*, *newpath*)

Moves a node at oldpath and all its descendant nodes to newpath. newpath must not exist

async **saveHiveTree**(*path=()*)

async **set**(*full*, *valu*, *nexs=False*)

A set operation at the hive level (full path).

async **storNodeDele**(*path*)

async **storNodeValu**(*full*, *valu*)

class synapse.lib.hive.**HiveApi**

Bases: *Base*

async **addAndSync**(*path*, *valu*, *iden*)

async **edits**()

async **get**(*full*)

async **loadHiveTree**(*tree*, *path=()*, *trim=False*)

async **popAndSync**(*path*, *iden*, *nexs=False*)

async **saveHiveTree**(*path=()*)

async **setAndSync**(*path*, *valu*, *iden*, *nexs=False*)

async **treeAndSync**(*path*, *iden*)

class synapse.lib.hive.**HiveDict**

Bases: *Base*

get(*name*, *default=None*)

items()

pack()

async pop(*name*, *default=None*)

async set(*name*, *valu*, *nexs=None*)

setdefault(*name*, *valu*)

values()

class synapse.lib.hive.**Node**

Bases: *Base*

A single node within the Hive tree.

async add(*valu*)

Increments existing node valu

async dict(*nexs=False*)

Get a HiveDict for this Node.

Returns
A HiveDict for this Node.

Return type
HiveDict

dir()

get(*name*)

name()

async open(*path*)

Open a child Node of the this Node.

Parameters
path (*tuple*) – A child path of the current node.

Returns
A Node at the child path.

Return type
Node

parent()

async pop(*path=()*)

async set(*valu*)

class synapse.lib.hive.**SlabHive**

Bases: *Hive*

async **storNodeDele**(*full*)

async **storNodeValu**(*full*, *valu*)

class synapse.lib.hive.**TeleHive**

Bases: *Hive*

A Hive that acts as a consistent read cache for a telepath proxy Hive

async **add**(*path*, *valu*)

Atomically increments a node's value.

async **get**(*path*)

Get the value of a node at a given path.

Parameters
full (*tuple*) – A full path tuple.

Returns
Arbitrary node value.

async **open**(*path*)

Open and return a hive Node().

Parameters
full (*tuple*) – A full path tuple.

Returns
A Hive node.

Return type
Node

async **pop**(*path*, *nexs=False*)

Remove and return the value for the given node.

async **set**(*path*, *valu*, *nexs=False*)

A set operation at the hive level (full path).

synapse.lib.hive.**iterpath**(*path*)

async synapse.lib.hive.**opendir**(*dirn*, *conf=None*)

async synapse.lib.hive.**openurl**(*url*, ***opts*)

synapse.lib.hiveauth module

class synapse.lib.hiveauth.**Auth**

Bases: *Pusher*

Auth is a user authentication and authorization stored in a Hive. Users correspond to separate logins with different passwords and potentially different privileges.

Users are assigned "rules". These rules are evaluated in order until a rule matches. Each rule is a tuple of boolean, and a rule path (a sequence of strings). Rules that are prefixes of a privilege match, i.e. a rule ('foo',) will match ('foo', 'bar').

Roles are just collections of rules. When a user is "granted" a role those rules are assigned to that user. Unlike in an RBAC system, users don't explicitly assume a role; they are merely a convenience mechanism to easily assign the same rules to multiple users.

Authgates are objects that manage their own authorization. Each AuthGate has roles and users subkeys which contain rules specific to that user or role for that AuthGate. The roles and users of an AuthGate, called GateRole and GateUser respectively, contain the iden of a role or user defined prior and rules specific to that role or user; they do not duplicate the metadata of the role or user.

Node layout:

```
Auth root (passed into constructor)
├ roles
│   ├ <role iden 1>
│   ├ ...
│   └ last role
├ users
│   ├ <user iden 1>
│   ├ ...
│   └ last user
└ authgates
    ├ <iden 1>
    │   ├ roles
    │   │   ├ <role iden 1>
    │   │   ├ ...
    │   │   └ last role
    │   └ users
    │       ├ <user iden 1>
    │       ├ ...
    │       └ last user
    ├ <iden 2>
    │   ├ ...
    └ ... last authgate
```

async **addAuthGate**(*iden*, *authgatetype*)

Retrieve AuthGate by iden. Create if not present.

Note: Not change distributed

Returns
(HiveAuthGate)

async **addRole**(*name*, *iden=None*)

async **addUser**(*name*, *passwd=None*, *email=None*, *iden=None*)

Add a User to the Hive.

Parameters

- **name** (*str*) – The name of the User.
- **passwd** (*str*) – A optional password for the user.
- **email** (*str*) – A optional email for the user.
- **iden** (*str*) – A optional iden to use as the user iden.

Returns
A Hive User.

Return type
HiveUser

`checkUserLimit()`

Check if we're at the specified user limit.

This should be called right before adding/unlocking/unarchiving a user.

Raises: s_exc.HitLimit if the number of active users is at the maximum.

`async delAuthGate(iden)`

Delete AuthGate by iden.

Note: Not change distributed

`async delRole(iden)`

`async delUser(iden)`

`async feedBeholder(evnt, info, gateiden=None, logged=True)`

`getAuthGate(iden)`

`getAuthGates()`

`async getRoleByName(name)`

`async getUserByName(name)`

Get a user by their username.

Parameters
`name` (*str*) – Name of the user to get.

Returns
A Hive User. May return None if there is no user by the requested name.

Return type
HiveUser

`async getUserIdenByName(name)`

`reqAuthGate(iden)`

`async reqRole(iden)`

`async reqRoleByName(name)`

`async reqUser(iden)`

`async reqUserByName(name)`

`async reqUserByNameOrIden(name)`

`role(iden)`

`roles()`

`async setRoleInfo(iden, name, valu, gateiden=None, logged=True, mesg=None)` async **setRoleName**(*iden*, *name*)

async **setUserInfo**(*iden*, *name*, *valu*, *gateiden=None*, *logged=True*, *mesg=None*)

async **setUserName**(*iden*, *name*)

user(*iden*)

users()

class synapse.lib.hiveauth.**AuthGate**

Bases: *Base*

The storage object for object specific rules for users/roles.

async **delete**()

async **genRoleInfo**(*iden*)

async **genUserInfo**(*iden*)

pack()

class synapse.lib.hiveauth.**HiveRole**

Bases: *HiveRuler*

A role within the Hive authorization subsystem.

A role in HiveAuth exists to bundle rules together so that the same set of rules can be applied to multiple users.

allowed(*perm*, *default=None*, *gateiden=None*)

clearAuthCache()

async **genGateInfo**(*gateiden*)

pack()

async **setName**(*name*)

class synapse.lib.hiveauth.**HiveRuler**

Bases: *Base*

A HiveNode that holds a list of rules. This includes HiveUsers, HiveRoles, and the AuthGate variants of those

async **addRule**(*rule*, *indx=None*, *gateiden=None*, *nexs=True*)

async **delRule**(*rule*, *gateiden=None*)

getRules(*gateiden=None*)

async **setRules**(*rules*, *gateiden=None*, *nexs=True*, *mesg=None*)

class synapse.lib.hiveauth.**HiveUser**

Bases: *HiveRuler*

A user (could be human or computer) of the system within HiveAuth.

Cortex-wide rules are stored here. AuthGate-specific rules for this user are stored in an GateUser.

async **allow**(*perm*)

allowed(*perm*, *default=None*, *gateiden=None*)

clearAuthCache()

confirm(*perm*, *default=None*, *gateiden=None*)

async **genGateInfo**(*gateiden*)

getAllowedReason(*perm*, *default=None*, *gateiden=None*)
A routine which will return a tuple of (allowed, info).

getRoles()

async **grant**(*roleiden*, *indx=None*)

hasRole(*iden*)

isAdmin(*gateiden=None*)

isArchived()

isLocked()

pack(*packroles=False*)

raisePermDeny(*perm*, *gateiden=None*)

reqAdmin(*gateiden=None*, *mesg=None*)

async **revoke**(*iden*, *nexs=True*)

async **setAdmin**(*admin*, *gateiden=None*, *logged=True*)

async **setArchived**(*archived*)

async **setLocked**(*locked*, *logged=True*)

async **setName**(*name*)

async **setPasswd**(*passwd*, *nexs=True*)

async **setRoles**(*roleidens*)
Replace all the roles for a given user with a new list of roles.

Parameters
roleidens (`list`) – A list of roleidens.

Notes

The roleiden for the "all" role must be present in the new list of roles. This replaces all existing roles that the user has with the new roles.

Returns
None

async **tryPasswd**(*passwd*, *nexs=True*)

synapse.lib.hiveauth.**getShadow**(*passwd*)
This API is deprecated.

synapse.lib.hiveauth.**textFromRule**(*rule*)

synapse.lib.httpapi module

class synapse.lib.httpapi.**ActiveV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async get()

class synapse.lib.httpapi.**AuthAddRoleV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.httpapi.**AuthAddUserV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.httpapi.**AuthDelRoleV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.httpapi.**AuthGrantV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

/api/v1/auth/grant?user=iden&role=iden

async get()

async post()

class synapse.lib.httpapi.**AuthRevokeV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

/api/v1/auth/grant?user=iden&role=iden

async get()

async post()

class synapse.lib.httpapi.**AuthRoleV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async get(*iden*)

async post(*iden*)

class synapse.lib.httpapi.**AuthRolesV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async get()

class synapse.lib.httpapi.**AuthUserPasswdV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async **post**(*iden*)

class synapse.lib.httpapi.**AuthUserV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async **get**(*iden*)

async **post**(*iden*)

class synapse.lib.httpapi.**AuthUsersV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async **get**()

class synapse.lib.httpapi.**BeholdSockV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *WebSocket*

async **onInitMessage**(*byts*)

async **on_message**(*byts*)

Handle incoming messages on the WebSocket

This method must be overridden.

Changed in version 4.5: `on_message` can be a coroutine.

class synapse.lib.httpapi.**CoreInfoV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

/api/v1/core/info

async **get**()

class synapse.lib.httpapi.**ExtApiHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

/api/ext/.*

compute_etag()

Computes the etag header to be used for this request.

By default uses a hash of the content written so far.

May be overridden to provide custom etag implementations, or may return None to disable tornado's default etag support.

async **delete**(*path*)

async **get**(*path*)

async **head**(*path*)

async **options**(*path*)

async **patch**(*path*)

async **post**(*path*)

async **put**(*path*)

set_default_headers()

Override this to set HTTP headers at the beginning of the request.

For example, this is the place to set a custom `Server` header. Note that setting such headers in the normal flow of request processing may not do what you want, since headers may be reset during error handling.

storm_prefix = 'init { $request = $lib.cortex.httpapi.response($_http_request_info) }'

class synapse.lib.httpapi.**FeedV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: `Handler`

/api/v1/feed

Examples

Example data:

```
{
    'name': 'syn.nodes',
    'view': null,
    'items': [...],
}
```

async **post**()

class synapse.lib.httpapi.**Handler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: `HandlerBase`, `RequestHandler`

on_connection_close()

Called in async handlers if the client closed the connection.

Override this to clean up resources associated with long-lived connections. Note that this method is called only if the connection was closed during asynchronous processing; if you need to do cleanup after every request override *on_finish* instead.

Proxies may keep a connection open for a time (perhaps indefinitely) after the client has gone away, so this method may not be called promptly after the end user closes their connection.

prepare()

Called at the beginning of a request before *get*/*post*/etc.

Override this method to perform common initialization regardless of the request method.

Asynchronous support: Use `async def` or decorate this method with *.gen.coroutine* to make it asynchronous. If this method returns an `Awaitable` execution will not proceed until the `Awaitable` is done.

New in version 3.1: Asynchronous support.

class synapse.lib.httpapi.**HandlerBase**

Bases: object

async **allowed**(*perm*, *default=False*, *gateiden=None*)

Check if the authenticated user has the given permission.

Parameters

- **perm** (*tuple*) – The permission tuple to check.
- **default** (*boolean*) – The default value for the permission.
- **gateiden** (*str*) – The gateiden to check the permission against.

Notes

This API sets up HTTP response values if it returns False.

Returns
True if the user has the requested permission.

Return type
bool

async **authenticated**()

Check if the request has an authenticated user or not.

Returns
True if the request has an authenticated user, false otherwise.

Return type
bool

check_origin(*origin*)

getAuthCell()

Return a reference to the cell used for auth operations.

getCustomHeaders()

getJsonBody(*validator=None*)

async **getUseridenBody**(*validator=None*)

Helper function to confirm that there is an auth user and a valid JSON body in the request.

Parameters
validator – Validator function run on the deserialized JSON body.

Returns
The user definition and body of the request as deserialized JSON, or a tuple of s_common.novalu objects if there was no user or json body.

Return type
(str, object)

async **handleApiKeyAuth**()

async **handleBasicAuth**()

Handle basic authentication in the handler.

Notes

Implementors may override this to disable or implement their own basic auth schemes. This is expected to set web_useriden and web_username upon successful authentication.

Returns
The user iden of the logged in user.

Return type
str

`initialize(cell)`

`isOrigHost(origin)`

`async isUserAdmin()`

Check if the current authenticated user is an admin or not.

Returns
True if the user is an admin, false otherwise.

Return type
bool

`loadJsonMesg(byts, validator=None)`

`logAuthIssue(mesg=None, user=None, username=None, level=30)`

Helper to log issues related to request authentication.

Parameters

- **mesg** (*str*) – Additional message to log.
- **user** (*str*) – User iden, if available.
- **username** (*str*) – Username, if available.
- **level** (*int*) – Logging level to log the message at. Defaults to logging.WARNING.

Returns
None

`options()`

`async reqAuthAdmin()`

Require the current authenticated user to be an admin.

Notes

If this returns False, an error message has already been sent and no additional processing for the request should be done.

Returns
True if the user is an admin, false otherwise.

Return type
bool

`async reqAuthUser()`

`sendAuthRequired()`

`sendRestErr(code, mesg)`

`sendRestExc(e)`

`sendRestRetn(valu)`

`async sess(gen=True)`

Get the heavy Session object for the request.

Parameters
gen (*bool*) – If set to True, generate a new session if there is no sess cookie.

Notes

This stores the identifier in the `sess` cookie for with a 14 day expiration, stored in the Cell. Valid requests with that `sess` cookie will resolve to the same Session object.

Returns
A heavy session object. If the sess cookie is invalid or gen is false, this returns None.

Return type
Sess

`set_default_headers()`

`async useriden()`

Get the user iden of the current session user.

Note: This function will pull the iden from the current session, or attempt to resolve the useriden with basic authentication.

Returns
The iden of the current session user.

Return type
str

`class synapse.lib.httpapi.HealthCheckV1(application: Application, request: HTTPServerRequest, **kwargs: Any)`

Bases: `Handler`

`async get()`

`class synapse.lib.httpapi.LoginV1(application: Application, request: HTTPServerRequest, **kwargs: Any)`

Bases: `Handler`

`async post()`

`class synapse.lib.httpapi.LogoutV1(application: Application, request: HTTPServerRequest, **kwargs: Any)`

Bases: `Handler`

`async get()`

class synapse.lib.httpapi.**ModelNormV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async **get**()

async **post**()

class synapse.lib.httpapi.**ModelV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async **get**()

class synapse.lib.httpapi.**OnePassIssueV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

/api/v1/auth/onepass/issue

async **post**()

class synapse.lib.httpapi.**ReqValidStormV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

async **get**()

async **post**()

class synapse.lib.httpapi.**RobotHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *HandlerBase*, RequestHandler

async **get**()

class synapse.lib.httpapi.**Sess**

Bases: *Base*

addWebSock(*sock*)

delWebSock(*sock*)

async **login**(*user*)

async **logout**()

async **set**(*name*, *valu*)

async **update**(*vals: dict*)

class synapse.lib.httpapi.**StormCallV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

async **get**()

async **post**()

class synapse.lib.httpapi.**StormExportV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

async get()

async post()

class synapse.lib.httpapi.**StormHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

getCore()

class synapse.lib.httpapi.**StormNodesV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

async get()

async post()

class synapse.lib.httpapi.**StormV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *StormHandler*

async get()

async post()

class synapse.lib.httpapi.**StormVarsGetV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async get()

class synapse.lib.httpapi.**StormVarsPopV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.httpapi.**StormVarsSetV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

async post()

class synapse.lib.httpapi.**StreamHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *Handler*

Subclass for Tornado streaming uploads.

Notes

- Async method prepare() is called after headers are read but before body processing.
- Sync method on_finish() can be used to cleanup after a request.
- Sync method on_connection_close() can be used to cleanup after a client disconnect.
- Async methods post(), put(), etc are called after the streaming has completed.

async **data_received**(*chunk*)

Implement this method to handle streamed request data.

Requires the *.stream_request_body* decorator.

May be a coroutine for flow control.

class synapse.lib.httpapi.**WebSocket**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *HandlerBase*, WebSocketHandler

async **xmit**(*name*, ***info*)

synapse.lib.ingest module

synapse.lib.interval module

A few utilities for dealing with intervals.

synapse.lib.interval.**fold**(**vals*)

Initialize a new (min,max) tuple interval from values.

Parameters
***vals** (*[int, ...]*) – A list of values (or Nones)

Returns
A (min,max) interval tuple or None

Return type
((int,int))

synapse.lib.interval.**overlap**(*ival0*, *ival1*)

Determine if two interval tuples have overlap.

Parameters

- **ival0** (*(int,int)*) – An interval tuple
- **ival1** (*(int,int)*) –

Returns
True if the intervals overlap, otherwise False

Return type
(bool)

synapse.lib.interval.**parsetime**(*text*)

Parse an interval time string and return a (min,max) tuple.

Parameters
text (*str*) – A time interval string

Returns
A epoch millis epoch time string

Return type
((int,int))

synapse.lib.jsonstor module

class synapse.lib.jsonstor.**JsonStor**

Bases: *Base*

A filesystem like storage mechanism that allows hirarchical lookup of reference counted "objects" that have individually editable properties.

#TODO json validation by path glob matches? (persists?) #TODO GUID ACCESS with index generation by type #TODO registered types jsonschema with optional write-back validation

async **cmpDelPathObjProp**(*path*, *prop*, *valu*)

async **copyPathObj**(*oldp*, *newp*)

async **copyPathObjs**(*paths*)

async **delPathObj**(*path*)

Remove a path and decref the object it references.

async **delPathObjProp**(*path*, *prop*)

async **getPathList**(*path*)

async **getPathObj**(*path*)

async **getPathObjProp**(*path*, *prop*)

async **getPathObjs**(*path*)

async **hasPathObj**(*path*)

async **popPathObjProp**(*path*, *prop*, *defv=None*)

async **setPathLink**(*srcpath*, *dstpath*)

Add a link from the given srcpath to the dstpath. NOTE: This causes the item at dstpath to be incref'd

async **setPathObj**(*path*, *item*)

Set (and/or reinitialize) the object at the given path.

NOTE: This will break any links by creating a new object.

async **setPathObjProp**(*path*, *prop*, *valu*)

class synapse.lib.jsonstor.**JsonStorApi**

Bases: *CellApi*

async **addQueue**(*name*, *info*)

addUserNotif(*useriden*, *mesgtype*, *mesgdata=None*)

async **cmpDelPathObjProp**(*path*, *name*, *valu*) async **copyPathObj**(*oldp*, *newp*)

async **copyPathObjs**(*paths*)

async **cullQueue**(*name*, *offs*)

async **delPathObj**(*path*)

async **delPathObjProp**(*path*, *name*)

async **delQueue**(*name*)

delUserNotif(*indx*)

async **getPathList**(*path*)

async **getPathObj**(*path*)

async **getPathObjProp**(*path*, *prop*)

async **getPathObjs**(*path*)

getUserNotif(*indx*)

async **getsQueue**(*name*, *offs*, *size=None*, *cull=True*, *wait=True*)

async **hasPathObj**(*path*)

iterUserNotifs(*useriden*, *size=None*)

async **popPathObjProp**(*path*, *prop*)

async **putsQueue**(*name*, *items*)

async **setPathLink**(*srcpath*, *dstpath*)

async **setPathObj**(*path*, *item*)

async **setPathObjProp**(*path*, *prop*, *valu*)

watchAllUserNotifs(*offs=None*)

class synapse.lib.jsonstor.**JsonStorCell**

Bases: *Cell*

async **addQueue**(*name*, *info*)

async **addUserNotif**(*useriden*, *mesgtype*, *mesgdata=None*)

cellapi

alias of *JsonStorApi*

async **cmpDelPathObjProp**(*path*, *name*, *valu*)

async **copyPathObj**(*oldp*, *newp*)

async **copyPathObjs**(*paths*)

async **cullQueue**(*name*, *offs*)

async **delPathObj**(*path*)

async **delPathObjProp**(*path*, *name*)

async **delQueue**(*name*)

async **delUserNotif**(*indx*)

classmethod **getEnvPrefix**()
Get a list of envar prefixes for config resolution.

async **getPathList**(*path*)

async **getPathObj**(*path*)

async **getPathObjProp**(*path*, *prop*)

async **getPathObjs**(*path*)

async **getUserNotif**(*indx*)

async **getsQueue**(*name*, *offs*, *size=None*, *cull=True*, *wait=True*)

async **hasPathObj**(*path*)

async **initServiceStorage**()

async **iterUserNotifs**(*useriden*, *size=None*)

async **popPathObjProp**(*path*, *prop*)

async **putsQueue**(*name*, *items*)

async **setPathLink**(*srcpath*, *dstpath*)

async **setPathObj**(*path*, *item*)

async **setPathObjProp**(*path*, *prop*, *valu*)

async **watchAllUserNotifs**(*offs=None*)

synapse.lib.jupyter module

class synapse.lib.jupyter.**CmdrCore**
Bases: *Base*

A helper for jupyter/cmdr CLI interaction

async **addFeedData**(*name*, *items*, *, *viewiden=None*)
Add feed data to the cortex.

async **eval**(*text*, *opts=None*, *num=None*, *cmdr=False*)
A helper for executing a storm command and getting a list of packed nodes.

Parameters

- **text** (*str*) – Storm command to execute.
- **opts** (*dict*) – Opt to pass to the cortex during execution.
- **num** (*int*) – Number of nodes to expect in the output query. Checks that with an assert statement.
- **cmdr** (*bool*) – If True, executes the line via the Cmdr CLI and will send output to outp.

Notes

The opts dictionary will not be used if cmdr=True.

Returns
A list of packed nodes.

Return type
list

`async runCmdLine(text)`

Run a line of text directly via cmdr.

`async storm(text, opts=None, num=None, cmdr=False, suppress_logging=False)`

A helper for executing a storm command and getting a list of storm messages.

Parameters

- **text** (*str*) – Storm command to execute.
- **opts** (*dict*) – Opt to pass to the cortex during execution.
- **num** (*int*) – Number of nodes to expect in the output query. Checks that with an assert statement.
- **cmdr** (*bool*) – If True, executes the line via the Cmdr CLI and will send output to outp.
- **suppress_logging** (*bool*) – If True, suppresses some logging related to Storm runtime exceptions.

Notes

The opts dictionary will not be used if cmdr=True.

Returns
A list of storm messages.

Return type
list

`suppress_logging(suppress)`

Context manager to suppress specific loggers.

`class synapse.lib.jupyter.StormCore`

Bases: *Base*

A helper for jupyter/storm CLI interaction

`async runCmdLine(text, opts=None)`

Run a line of text directly via storm cli.

`async storm(text, opts=None, num=None, cli=False, suppress_logging=False)`

A helper for executing a storm command and getting a list of storm messages.

Parameters

- **text** (*str*) – Storm command to execute.
- **opts** (*dict*) – Opt to pass to the cortex during execution.
- **num** (*int*) – Number of nodes to expect in the output query. Checks that with an assert statement.

- **cli** (*bool*) – If True, executes the line via the Storm CLI and will send output to outp.
- **suppress_logging** (*bool*) – If True, suppresses some logging related to Storm runtime exceptions.

Notes

The opts dictionary will not be used if cmdr=True.

Returns
A list of storm messages.

Return type
list

suppress_logging(*suppress*)

Context manager to suppress specific loggers.

synapse.lib.jupyter.**genTempCoreProxy**(*mods=None*)

Get a temporary cortex proxy.

synapse.lib.jupyter.**genTempStormsvcProxy**(*cmdrcore*, *svcname*, *svcctor*, *conf=None*)

synapse.lib.jupyter.**getDocData**(*fp*, *root=None*)

Parameters

- **fp** (*str*) – Name of the file to retrieve the data of.
- **root** (*str*) – Optional root path to look for a docdata directory in.

Notes

Will detect json/jsonl/yaml/mpk extensions and automatically decode that data if found; otherwise it returns bytes.

Defaults to looking for the `docdata` directory in the current working directory. This behavior works fine for notebooks nested in the docs directory of synapse; but this root directory that is looked for may be overridden by providing an alternative root.

Returns
May be deserialized data or bytes.

Return type
data

Raises
ValueError if the file does not exist or directory traversal attempted..
–

synapse.lib.jupyter.**getDocPath**(*fn*, *root=None*)

Helper for getting a documentation data file paths.

Parameters

- **fn** (*str*) – Name of the file to retrieve the full path for.
- **root** (*str*) – Optional root path to look for a docdata in.

Notes

Defaults to looking for the `docdata` directory in the current working directory. This behavior works fine for notebooks nested in the docs directory of synapse; but this root directory that is looked for may be overridden by providing an alternative root.

Returns
A file path.

Return type
str

Raises
ValueError if the file does not exist or directory traversal attempted.. –

async synapse.lib.jupyter.**getItemCmdr**(*prox*, *outp=None*, *locs=None*)

Get a Cmdr instance with prepopulated locs

async synapse.lib.jupyter.**getItemStorm**(*prox*, *outp=None*)

Get a Storm CLI instance with prepopulated locs

async synapse.lib.jupyter.**getTempCoreCmdr**(*mods=None*, *outp=None*)

Get a CmdrCore instance which is backed by a temporary Cortex.

Parameters

- **mods** (*list*) – A list of additional CoreModules to load in the Cortex.
- **outp** – A output helper. Will be used for the Cmdr instance.

Notes

The CmdrCore returned by this should be fini()'d to tear down the temporary Cortex.

Returns
A CmdrCore instance.

Return type
CmdrCore

async synapse.lib.jupyter.**getTempCoreCmdrStormsvc**(*svcname*, *svcctor*, *svcconf=None*, *outp=None*)

Get a proxy to a Storm service and a CmdrCore instance backed by a temporary Cortex with the service added.

Parameters

- **svcname** (*str*) – Storm service name
- **svcctor** – Storm service constructor (e.g. Example.anit)
- **svcconf** – Optional conf for the Storm service
- **outp** – A output helper for the Cmdr instance

Notes

Both the CmdrCore and Storm service proxy should be fini()'d for proper teardown

Returns
A CmdrCore instance and proxy to the Storm service

Return type
(*CmdrCore*, *Proxy*)

`async synapse.lib.jupyter.getTempCoreProx(mods=None)`

Get a Telepath Proxt to a Cortex instance which is backed by a temporary Cortex.

Parameters
mods (*list*) – A list of additional CoreModules to load in the Cortex.

Notes

The Proxy returned by this should be fini()'d to tear down the temporary Cortex.

Returns
s_telepath.Proxy

`async synapse.lib.jupyter.getTempCoreStorm(mods=None, outp=None)`

Get a StormCore instance which is backed by a temporary Cortex.

Parameters

- **mods** (*list*) – A list of additional CoreModules to load in the Cortex.
- **outp** – A output helper. Will be used for the Cmdr instance.

Notes

The StormCore returned by this should be fini()'d to tear down the temporary Cortex.

Returns
A StormCore instance.

Return type
StormCore

`async synapse.lib.jupyter.getTempCoreStormStormsvc(svcname, svcctor, svcconf=None, outp=None)`

Get a proxy to a Storm service and a StormCore instance backed by a temporary Cortex with the service added.

Parameters

- **svcname** (*str*) – Storm service name
- **svcctor** – Storm service constructor (e.g. Example.anit)
- **svcconf** – Optional conf for the Storm service
- **outp** – A output helper for the Cmdr instance

Notes

Both the StormCore and Storm service proxy should be fini()'d for proper teardown

Returns
A StormCore instance and proxy to the Storm service

Return type
(*StormCore*, *Proxy*)

synapse.lib.jupyter.**suppress_logging**(*suppress*)
Context manager to suppress specific loggers.

synapse.lib.layer module

The Layer 2.0 archtecture introduces several optimized node/message serialization formats used by the layers to optimize returning primitives and facilitate efficient node construction:

Note: This interface is subject to change between minor revisions.

Storage Types (<stortype>)

In Layers 2.0, each node property from the model has an associated "storage type". Each storage type determines how the data is indexed and represented within the Layer. This formalizes the separation of "data model" from "storage model". Each data model type has a "stortype" property which coresponds to one of the STOR_TYPE_XXX values. The knowledge of the mapping of data model types to storage types is the responsibility of the data model, making the Layer implementation fully decoupled from the data model.

Node Edits / Edits

A node edit consists of a (<buid>, <form>, [edits]) tuple. An edit is Tuple of (<type>, <info>, List[NodeEdits]) where the first element is an int that matches to an EDIT_* constant below, the info is a tuple that varies depending on the first element, and the third element is a list of dependent NodeEdits that will only be applied if the edit actually makes a change.

Storage Node (<sode>)

A storage node is a layer/storage optimized node representation which is similar to a "packed node". A storage node *may* be partial (as it is produced by a given layer) and are joined by the view/snap into "full" storage nodes which are used to construct Node() instances.

Sode format:

```
(<buid>, {

    'ndef': (<formname>, <formvalu>),

    'props': {
        <propname>: <propvalu>,
    }

    'tags': {
        <tagname>: <tagvalu>,
    }

```

(continues on next page)

(continued from previous page)

```
    'tagprops: {
        <tagname>: {
            <propname>: <propvalu>,
        },
    }

    # changes that were *just* made.
    'edits': [
        <edit>
    ]

}),
```

class synapse.lib.layer.**IndxBy**(*layr*, *abrv*, *db*)

Bases: object

IndxBy sub-classes encapsulate access methods and encoding details for various types of properties within the layer to be lifted/compared by storage types.

buidsByDups(*indx*)

buidsByPref(*indx=b''*)

buidsByRange(*minindx*, *maxindx*)

buidsByRangeBack(*minindx*, *maxindx*)

getNodeValu(*buid*)

hasIndxBuid(*indx*, *buid*)

keyBuidsByDups(*indx*)

keyBuidsByDupsBack(*indx*)

keyBuidsByPref(*indx=b''*)

keyBuidsByPrefBack(*indx=b''*)

keyBuidsByRange(*minindx*, *maxindx*)

keyBuidsByRangeBack(*minindx*, *maxindx*)

Yields backwards from maxindx to minindx

scanByDups(*indx*)

scanByPref(*indx=b''*)

scanByPrefBack(*indx=b''*)

scanByRange(*minindx*, *maxindx*)

scanByRangeBack(*minindx*, *maxindx*)

class synapse.lib.layer.**IndxByForm**(*layr*, *form*)

Bases: *IndxBy*

getNodeValu(*buid*)

class synapse.lib.layer.**IndxByProp**(*layr, form, prop*)

Bases: *IndxBy*

getNodeValu(*buid*)

class synapse.lib.layer.**IndxByPropArray**(*layr, form, prop*)

Bases: *IndxBy*

getNodeValu(*buid*)

class synapse.lib.layer.**IndxByTag**(*layr, form, tag*)

Bases: *IndxBy*

getNodeValuForm(*buid*)

class synapse.lib.layer.**IndxByTagProp**(*layr, form, tag, prop*)

Bases: *IndxBy*

getNodeValu(*buid*)

class synapse.lib.layer.**Layer**

Bases: *Pusher*

The base class for a cortex layer.

async **clone**(*newdirn*)

Copy the contents of this layer to a new layer

async **delete**()

Delete the underlying storage

getAbrvProp(*abrv*)

async **getEdgeVerbs**()

async **getEdges**(*verb=None*)

async **getEditIndx**()

Returns what will be the *next* (i.e. 1 past the last) nodeedit log index.

async **getEditOffs**()

Return the offset of the last *recorded* log entry. Returns -1 if nodeedit log is disabled or empty.

async **getEditSize**()

async **getFormCounts**()

getFormProps()

getIdenFutu(*iden=None*)

async **getLayerSize**()

Get the total storage size for the layer.

async **getMirrorStatus**()

async **getModelVers**()

async getNodeData(*buid*, *name*)
Return a single element of a buid's node data

getNodeEditWindow()

async getNodeForm(*buid*)

async getNodeTag(*buid*, *tag*)

async getNodeValu(*buid*, *prop=None*)
Retrieve either the form valu or a prop valu for the given node by buid.

getPropAbrv(*form*, *prop*)

async getPropArrayCount(*formname*, *propname=None*)
Return the number of invidiual value rows in the layer for the given array form/prop.

getPropArrayValuCount(*formname*, *propname*, *stortype*, *valu*)

async getPropCount(*formname*, *propname=None*, *maxsize=None*)
Return the number of property rows in the layer for the given form/prop.

getPropValuCount(*formname*, *propname*, *stortype*, *valu*)

getStorIndx(*stortype*, *valu*)

async getStorNode(*buid*)

getStorNodeCount()

async getStorNodes()
Yield (buid, sode) tuples for all the nodes with props/tags/tagprops stored in this layer.

async getTagCount(*tagname*, *formname=None*)
Return the number of tag rows in the layer for the given tag/form.

getTagPropAbrv(**args*)

async getTagPropCount(*form*, *tag*, *prop*)
Return the number of property rows in the layer for the given form/tag/prop.

getTagPropValuCount(*form*, *tag*, *prop*, *stortype*, *valu*)

getTagProps()

async getUnivPropCount(*propname*, *maxsize=None*)
Return the number of universal property rows in the layer for the given prop.

async hasNodeData(*buid*, *name*)

async hasNodeEdge(*buid1*, *verb*, *buid2*)

async hasTagProp(*name*)

async initLayerActive()

async initLayerPassive()

async initUpstreamSync(*url*)

async iterEdgeVerbs(*n1buid*, *n2buid*)

async iterFormRows(*form*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes of a single form, optionally (re)starting at startvalu.

Parameters

- **form** (*str*) – A form name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns

AsyncIterator[Tuple(buid, valu)]

async iterLayerNodeEdits()

Scan the full layer and yield artificial sets of nodeedits.

async iterNodeData(*buid*)

Return a generator of all a buid's node data

async iterNodeDataKeys(*buid*)

Return a generator of all a buid's node data keys

async iterNodeEdgesN1(*buid*, *verb=None*)

async iterNodeEdgesN2(*buid*, *verb=None*)

async iterNodeEditLog(*offs=0*)

Iterate the node edit log and yield (offs, edits, meta) tuples.

async iterNodeEditLogBack(*offs=0*)

Iterate the node edit log and yield (offs, edits, meta) tuples in reverse.

async iterPropRows(*form*, *prop*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes with a particular secondary property, optionally (re)starting at startvalu.

Parameters

- **form** (*str*) – A form name.
- **prop** (*str*) – A universal property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns

AsyncIterator[Tuple(buid, valu)]

async iterTagPropRows(*tag*, *prop*, *form=None*, *stortype=None*, *startvalu=None*)

Yields (buid, valu) that match a tag:prop, optionally (re)starting at startvalu.

Parameters

- **tag** (*str*) – tag name
- **prop** (*str*) – prop name
- **form** (*Optional[str]*) – optional form name

- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

async **iterTagRows**(*tag*, *form=None*, *starttupl=None*)

Yields (buid, (valu, form)) values that match a tag and optional form, optionally (re)starting at starttupl.

Parameters

- **tag** (*str*) – the tag to match
- **form** (*Optional[str]*) – if present, only yields buids of nodes that match the form.
- **starttupl** (*Optional[Tuple[buid, form]]*) – if present, (re)starts the stream of values there.

Returns
AsyncIterator[Tuple(buid, (valu, form))]

Note: This yields (buid, (tagvalu, form)) instead of just buid, valu in order to allow resuming an interrupted call by feeding the last value retrieved into starttupl

---

async **iterUnivRows**(*prop*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes with a particular universal property, optionally (re)starting at startvalu.

Parameters

- **prop** (*str*) – A universal property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

async **iterWipeNodeEdits**()

async **liftByDataName**(*name*)

async **liftByFormValu**(*form*, *cmprvals*, *reverse=False*)

async **liftByProp**(*form*, *prop*, *reverse=False*)

async **liftByPropArray**(*form*, *prop*, *cmprvals*, *reverse=False*)

async **liftByPropValu**(*form*, *prop*, *cmprvals*, *reverse=False*)

async **liftByTag**(*tag*, *form=None*, *reverse=False*)

async **liftByTagProp**(*form*, *tag*, *prop*, *reverse=False*)

async **liftByTagPropValu**(*form*, *tag*, *prop*, *cmprvals*, *reverse=False*)

Note: form may be None

async **liftByTagValu**(*tag*, *cmpr*, *valu*, *form=None*, *reverse=False*) async **liftTagProp**(*name*)

mayDelBuid(*buid*, *sode*)

nodeeditctor

alias of *SlabSeqn*

async **pack**()

async **saveNodeEdits**(*edits*, *meta*)

Save node edits to the layer and return a tuple of (nexsoffs, changes).

Note: nexsoffs will be None if there are no changes.

async **setLayerInfo**(*name*, *valu*)

async **setModelVers**(*vers*)

setPropAbrv(*form*, *prop*)

setSodeDirty(*buid*, *sode*, *form*)

setTagPropAbrv(**args*)

async **stat**()

async **storNodeEdits**(*nodeedits*, *meta*)

async **storNodeEditsNoLift**(*nodeedits*, *meta*)

Execute a series of node edit operations.

Does not return the updated nodes.

async **syncIndexEvents**(*offs*, *matchdef*, *wait=True*)

Yield (offs, (buid, form, ETYPE, VALS, META)) tuples from the nodeedit log starting from the given offset. Only edits that match the filter in matchdef will be yielded.

Notes

ETYPE is an constant EDIT_* above. VALS is a tuple whose format depends on ETYPE, outlined in the comment next to the constant. META is a dict that may contain keys 'user' and 'time' to represent the iden of the user that initiated the change, and the time that it took place, respectively.

Additionally, every 1000 entries, an entry (offs, (None, None, EDIT_PROGRESS, (), ())) message is emitted.

The matchdef dict may contain the following keys: forms, props, tags, tagprops. The value must be a sequence of strings. Each key/val combination is treated as an "or", so each key and value yields more events. forms: EDIT_NODE_ADD and EDIT_NODE_DEL events. Matches events for nodes with forms in the value list. props: EDIT_PROP_SET and EDIT_PROP_DEL events. Values must be in form:prop or .universal form tags: EDIT_TAG_SET and EDIT_TAG_DEL events. Values must be the raw tag with no #. tagprops: EDIT_TAGPROP_SET and EDIT_TAGPROP_DEL events. Values must be just the prop or tag:prop.

Will not yield any values if this layer was not created with logedits enabled

Parameters

- **offs** (*int*) – starting nexus/editlog offset

- **matchdef** (Dict[*str*, *Sequence*[*str*]]) – a dict describing which events are yielded
- **wait** (*bool*) – whether to pend and stream value until this layer is fini'd

async **syncNodeEdits**(*offs*, *wait=True*)

Identical to syncNodeEdits2, but doesn't yield meta

async **syncNodeEdits2**(*offs*, *wait=True*)

Once caught up with storage, yield them in realtime.

Returns

Tuple of offset(int), nodeedits, meta(dict)

async **verify**(*config=None*)

async **verifyAllBuids**(*scanconf=None*)

async **verifyAllProps**(*scanconf=None*)

async **verifyAllTagProps**(*scanconf=None*)

async **verifyAllTags**(*scanconf=None*)

async **verifyBuidTag**(*buid*, *formname*, *tagname*, *tagvalu*)

async **verifyByBuid**(*buid*, *sode*)

async **verifyByProp**(*form*, *prop*, *autofix=None*)

async **verifyByPropArray**(*form*, *prop*, *autofix=None*)

async **verifyByTag**(*tag*, *autofix=None*)

async **verifyByTagProp**(*form*, *tag*, *prop*, *autofix=None*)

async **waitEditOffs**(*offs*, *timeout=None*)

Wait for the node edit log to write an entry at/past the given offset.

async **waitForHot**()

Wait for the layer's slab to be prefaulted and locked into memory if lockmemory is true, otherwise return.

async **waitUpstreamOffs**(*iden*, *offs*)

class synapse.lib.layer.**LayerApi**

Bases: *CellApi*

async **getEditIndx**()

Returns what will be the *next* nodeedit log index.

async **getEditSize**()

Return the total number of (edits, meta) pairs in the layer changelog.

async **getIden**()

async **iterLayerNodeEdits**()

Scan the full layer and yield artificial nodeedit sets.

saveNodeEdits(*edits*, *meta*)

Save node edits to the layer and return a tuple of (nexsoffs, changes).

Note: nexsoffs will be None if there are no changes.

async **storNodeEdits**(*nodeedits*, *meta=None*)

async **storNodeEditsNoLift**(*nodeedits*, *meta=None*)

async **syncNodeEdits**(*offs*, *wait=True*)

Yield (offs, nodeedits) tuples from the nodeedit log starting from the given offset.

Once caught up with storage, yield them in realtime.

async **syncNodeEdits2**(*offs*, *wait=True*)

class synapse.lib.layer.**StorType**(*layr*, *stortype*)

Bases: `object`

decodeIndx(*valu*)

indx(*valu*)

async **indxBy**(*liftby*, *cmpr*, *valu*, *reverse=False*)

async **indxByForm**(*form*, *cmpr*, *valu*, *reverse=False*)

async **indxByProp**(*form*, *prop*, *cmpr*, *valu*, *reverse=False*)

async **indxByPropArray**(*form*, *prop*, *cmpr*, *valu*, *reverse=False*)

async **indxByTagProp**(*form*, *tag*, *prop*, *cmpr*, *valu*, *reverse=False*)

async **verifyBuidProp**(*buid*, *form*, *prop*, *valu*)

class synapse.lib.layer.**StorTypeFloat**(*layr*, *stortype*, *size=8*)

Bases: *StorType*

FloatPackNegMax = b'\x80\x00\x00\x00\x00\x00\x00\x00'

FloatPackNegMin = b'\xff\xf0\x00\x00\x00\x00\x00\x00'

FloatPackPosMax = b'\x7f\xf0\x00\x00\x00\x00\x00\x00'

FloatPackPosMin = b'\x00\x00\x00\x00\x00\x00\x00\x00'

FloatPacker = <_struct.Struct object>

decodeIndx(*bytz*)

fpack()

S.pack(v1, v2, …) -> bytes

Return a bytes object containing values v1, v2, … packed according to the format string S.format. See help(struct) for more on format strings.

indx(*valu*)

class synapse.lib.layer.**StorTypeFqdn**(*layr*)

Bases: *StorTypeUtf8*

decodeIndx(*bytz*)

indx(*norm*)

class synapse.lib.layer.**StorTypeGuid**(*layr*)

Bases: *StorType*

decodeIndx(*bytz*)

indx(*valu*)

class synapse.lib.layer.**StorTypeHier**(*layr*, *stortype*, *sepr='.'*)

Bases: *StorType*

decodeIndx(*bytz*)

getHierIndx(*valu*)

indx(*valu*)

class synapse.lib.layer.**StorTypeHugeNum**(*layr*, *stortype*)

Bases: *StorType*

decodeIndx(*bytz*)

getHugeIndx(*norm*)

indx(*norm*)

class synapse.lib.layer.**StorTypeInt**(*layr*, *stortype*, *size*, *signed*)

Bases: *StorType*

decodeIndx(*bytz*)

getIntIndx(*valu*)

indx(*valu*)

class synapse.lib.layer.**StorTypeIpv6**(*layr*)

Bases: *StorType*

decodeIndx(*bytz*)

getIPv6Indx(*valu*)

indx(*valu*)

class synapse.lib.layer.**StorTypeIval**(*layr*)

Bases: *StorType*

decodeIndx(*bytz*)

indx(*valu*)

class synapse.lib.layer.**StorTypeLatLon**(*layr*)

Bases: *StorType*

decodeIndx(*bytz*)

indx(*valu*)

class synapse.lib.layer.**StorTypeLoc**(*layr*)

Bases: *StorTypeHier*

class synapse.lib.layer.**StorTypeMsgp**(*layr*)

Bases: *StorType*

indx(*valu*)

class synapse.lib.layer.**StorTypeTag**(*layr*)

Bases: *StorTypeHier*

static getTagFilt(*cmpr*, *valu*)

class synapse.lib.layer.**StorTypeTime**(*layr*)

Bases: *StorTypeInt*

class synapse.lib.layer.**StorTypeUtf8**(*layr*)

Bases: *StorType*

decodeIndx(*bytz*)

indx(*valu*)

synapse.lib.layer.**getFlatEdits**(*nodeedits*)

synapse.lib.layer.**getNodeEditPerms**(*nodeedits*)

Yields (offs, perm) tuples that can be used in user.allowed()

synapse.lib.link module

class synapse.lib.link.**Link**

Bases: *Base*

A Link() is created to wrap a socket reader/writer.

feed(*byts*)

Used by Plex() to unpack bytes.

get(*name*, *defval=None*)

Get a property from the Link info.

getAddrInfo()

Get a summary of address information related to the link.

async getSpawnInfo()

getTlsPeerCn()

async recv(*size*)

async recvsize(*size*)

async rx()

async send(*byts*)

set(*name*, *valu*)

Set a property in the Link info.

async tx(*mesg*)

Async transmit routine which will wait for writer drain().

txfini()

async synapse.lib.link.**connect**(*host*, *port*, *ssl=None*, *hostname=None*, *linkinfo=None*)

Async connect and return a Link().

async synapse.lib.link.**fromspawn**(*spawninfo*)

async synapse.lib.link.**linkfile**(*mode='wb'*)

Connect a socketpair to a file-object and return (link, file).

async synapse.lib.link.**linksock**(*forceclose=False*)

Connect a Link, socket pair.

async synapse.lib.link.**listen**(*host*, *port*, *onlink*, *ssl=None*)

Listen on the given host/port and fire onlink(Link).

Returns a server object that contains the listening sockets

async synapse.lib.link.**unixconnect**(*path*)

Connect to a PF_UNIX server listening on the given path.

async synapse.lib.link.**unixlisten**(*path*, *onlink*)

Start an PF_UNIX server listening on the given path.

synapse.lib.lmdbslab module

class synapse.lib.lmdbslab.**GuidStor**(*slab*, *name*)

Bases: `object`

async **del_**(*iden*)

async **dict**(*iden*)

gen(*iden*)

has(*iden*)

set(*iden*, *name*, *valu*)

class synapse.lib.lmdbslab.**Hist**(*slab*, *name*)

Bases: `object`

A class for storing items in a slab by time.

Each added item is inserted into the specified db within the slab using the current epoch-millis time stamp as the key.

add(*item*, *tick=None*)

carve(*tick*, *tock=None*)

class synapse.lib.lmdbslab.**HotCount**

Bases: *HotKeyVal*

Like HotKeyVal, but optimized for integer/count vals

static **DecFunc**(*b*)

Decode a signed 64-bit int from 8 byte big-endian

static EncFunc(*i*)

Encode a signed 64-bit int into 8 byte big-endian bytes

get(*name: str*, *defv=0*)

inc(*name: str*, *valu=1*)

set(*name: str*, *valu*)

class synapse.lib.lmdbslab.**HotKeyVal**

Bases: *Base*

A hot-loop capable keyval that only syncs on commit.

static DecFunc(*byts*, *use_list=False*)

Use msgpack to de-serialize a python object.

Parameters

byts (*bytes*) – The bytes to de-serialize

Notes

String objects are decoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow decoding bad input strings.

Returns

The de-serialized object

Return type

obj

static EncFunc(*item*)

Use msgpack to serialize a compatible python object.

Parameters

item (*obj*) – The object to serialize

Notes

String objects are encoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow encoding bad input strings.

Returns

The serialized bytes in msgpack format.

Return type

bytes

delete(*name: str*)

get(*name: str*, *defv=None*)

pack()

set(*name: str*, *valu*)

sync()

class synapse.lib.lmdbslab.**LmdbBackup**

Bases: *Base*

async **saveto**(*dstdir*)

class synapse.lib.lmdbslab.**MultiQueue**

Bases: *Base*

Allows creation/consumption of multiple durable queues in a slab.

async **add**(*name*, *info*)

async **cull**(*name*, *offs*)

Remove up-to (and including) the queue entry at offs.

async **dele**(*name*, *minoffs*, *maxoffs*)

Remove queue entries from minoffs, up-to (and including) the queue entry at maxoffs.

exists(*name*)

async **get**(*name*, *offs*, *wait=False*, *cull=True*)

Return (nextoffs, item) tuple or (-1, None) for the given offset.

async **gets**(*name*, *offs*, *size=None*, *cull=False*, *wait=False*)

Yield (offs, item) tuples from the message queue.

list()

offset(*name*)

async **pop**(*name*, *offs*)

Pop a single entry from the named queue by offset.

async **put**(*name*, *item*, *reqid=None*)

async **puts**(*name*, *items*, *reqid=None*)

async **rem**(*name*)

async **sets**(*name*, *offs*, *items*)

Overwrite queue entries with the values in items, starting at offs.

size(*name*)

status(*name*)

class synapse.lib.lmdbslab.**Scan**(*slab*, *db*)

Bases: object

A state-object used by Slab. Not to be instantiated directly.

Parameters

- **slab** (Slab) – which slab the scan is over
- **db** (*str*) – name of open database on the slab

bump()

first()

isatitem()

Returns if the cursor is at the value in atitem

iterfunc()

iternext()

resume()

set_key(*lkey*)

set_range(*lkey*, *valu=None*)

class synapse.lib.lmdbslab.**ScanBack**(*slab*, *db*)

Bases: *Scan*

A state-object used by Slab. Not to be instantiated directly.

Scans backwards.

first()

iterfunc()

resume()

set_key(*lkey*)

set_range(*lkey*)

class synapse.lib.lmdbslab.**ScanKeys**(*slab*, *db*)

Bases: *Scan*

An iterator over the keys of the database. If the database is dupsort, a key with multiple values with be yielded once for each value.

isatitem()

Returns if the cursor is at the value in atitem

iterfunc()

iternext()

resume()

class synapse.lib.lmdbslab.**Slab**

Bases: *Base*

A "monolithic" LMDB instance for use in a asyncio loop thread.

COMMIT_PERIOD = 0.2

DEFAULT_GROWSIZE = None

DEFAULT_MAPSIZE = 1073741824

WARN_COMMIT_TIME_MS = 1000

addResizeCallback(*callback*)

allslabs = {}

`copydb`(*sourcedbname*, *destslab*, *destdbname=None*, *progresscb=None*)

Copy an entire database in this slab to a new database in potentially another slab.

Parameters

- **sourcedbname** (*str*) – name of the db in the source environment
- **destslab** (*LmdbSlab*) – which slab to copy rows to
- **destdbname** (*str*) – the name of the database to copy rows to in destslab
- **progresscb** (*Callable[int]*) – if not None, this function will be periodically called with the number of rows completed

Returns

the number of rows copied

Return type

(int)

Note: If any rows already exist in the target database, this method returns an error. This means that one cannot use destdbname=None unless there are no explicit databases in the destination slab.

async `copyslab`(*dstpath*, *compact=True*)

`count`(*lkey*, *db=None*)

async `countByPref`(*byts*, *db=None*, *maxsize=None*)

Return the number of rows in the given db with the matching prefix bytes.

`dbexists`(*name*)

The DB exists already if there's a key in the default DB with the name of the database

`delete`(*lkey*, *val=None*, *db=None*)

`dropdb`(*name*)

Deletes an **entire database** (i.e. a table), losing all data.

async `fini`()

Shut down the object and notify any onfini() coroutines.

Returns

Remaining ref count

`firstkey`(*db=None*)

Return the first key or None from the given db.

`forcecommit`()

Note: This method may raise a MapFullError

`get`(*lkey*, *db=None*)

async `getHotCount`(*name*)

async `getMultiQueue`(*name*, *nexsroot=None*) `getNameAbrv`(*name*)

`getSeqn`(*name*)

`async classmethod getSlabStats`()

`classmethod getSlabsInDir`(*dirn*)

Returns all open slabs under a directory

`has`(*lkey*, *db=None*)

`hasdup`(*lkey*, *lval*, *db=None*)

`async classmethod initSyncLoop`(*inst*)

`initdb`(*name*, *dupsort=False*, *integerkey=False*, *dupfixed=False*)

`last`(*db=None*)

Return the last key/value pair from the given db.

`lastkey`(*db=None*)

Return the last key or None from the given db.

`pop`(*lkey*, *db=None*)

`prefexists`(*byts*, *db=None*)

Returns True if a prefix exists in the db.

`put`(*lkey*, *lval*, *dupdata=False*, *overwrite=True*, *append=False*, *db=None*)

`putmulti`(*kvpairs*, *dupdata=False*, *append=False*, *db=None*)

Returns

Tuple of number of items consumed, number of items added

`rangeexists`(*lmin*, *lmax=None*, *db=None*)

Returns True if at least one key exists in the range.

`replace`(*lkey*, *lval*, *db=None*)

Like put, but returns the previous value if existed

`scanByDups`(*lkey*, *db=None*)

`scanByDupsBack`(*lkey*, *db=None*)

`scanByFull`(*db=None*)

`scanByFullBack`(*db=None*)

`scanByPref`(*byts*, *startkey=None*, *startvalu=None*, *db=None*)

Parameters

- **byts** (*bytes*) – prefix to match on
- **startkey** (*Optional[bytes]*) – if present, will start scanning at key=byts+startkey
- **startvalu** (*Optional[bytes]*) – if present, will start scanning at (key+startkey, startvalu)

Notes

startvalu only makes sense if byts+startkey matches an entire key. startvalu is only value for dupsort=True dbs

`scanByPrefBack(byts, db=None)`

`scanByRange(lmin, lmax=None, db=None)`

`scanByRangeBack(lmax, lmin=None, db=None)`

`scanKeys(db=None)`

`scanKeysByPref(byts, db=None)`

`stat(db=None)`

`statinfo()`

`async sync()`

`async classmethod syncLoopOnce()`

`async classmethod syncLoopTask()`

`syncevnt = None`

`synctask = None`

`async trash()`

Deletes underlying storage

`class synapse.lib.lmdbslab.SlabAbrv(slab, name)`

Bases: `object`

A utility for translating arbitrary bytes into fixed with id bytes

`abrvToByts(abrv)`

`abrvToName(byts)`

`bytsToAbrv(byts)`

`keys()`

`nameToAbrv(name)`

`names()`

`setBytsToAbrv(byts)`

`class synapse.lib.lmdbslab.SlabDict(slab, db=None, pref=b'')`

Bases: `object`

A dictionary-like object which stores its props in a slab via a prefix.

It is assumed that only one SlabDict with a given prefix exists at any given time, but it is up to the caller to cache them.

`get`(*name*, *defval=None*)

Get a name from the SlabDict.

Parameters

- **name** (*str*) – The key name.
- **defval** (*obj*) – The default value to return.

Returns
The return value, or None.

Return type
(obj)

`inc`(*name*, *valu=1*)

`items`()

Return a tuple of (prop, valu) tuples from the SlabDict.

Returns
Tuple of (name, valu) tuples.

Return type
(((str, object), …))

`keys`()

`pop`(*name*, *defval=None*)

Pop a name from the SlabDict.

Parameters

- **name** (*str*) – The name to remove.
- **defval** (*obj*) – The default value to return if the name is not present.

Returns
The object stored in the SlabDict, or defval if the object was not present.

Return type
object

`set`(*name*, *valu*)

Set a name in the SlabDict.

Parameters

- **name** (*str*) – The key name.
- **valu** (*obj*) – A msgpack compatible value.

Returns
None

synapse.lib.modelrev module

class synapse.lib.modelrev.**ModelRev**(*core*)

Bases: object

async **revCoreLayers**()

async **revModel20210126**(*layers*)

async **revModel20210312**(*layers*)

async **revModel20210528**(*layers*)

async **revModel20210801**(*layers*)

async **revModel20211112**(*layers*)

async **revModel20220307**(*layers*)

async **revModel20220315**(*layers*)

async **revModel20220509**(*layers*)

async **revModel20220706**(*layers*)

async **revModel20220803**(*layers*)

async **revModel20220901**(*layers*)

async **revModel20221025**(*layers*)

async **revModel20221123**(*layers*)

async **revModel20221212**(*layers*)

async **revModel20221220**(*layers*)

async **revModel20230209**(*layers*)

async **revModel_0_2_18**(*layers*)

async **revModel_0_2_19**(*layers*)

async **revModel_0_2_20**(*layers*)

async **revModel_0_2_21**(*layers*)

async **revModel_0_2_22**(*layers*)

async **revModel_0_2_23**(*layers*)

async **revModel_0_2_24**(*layers*)

async **runStorm**(*text*, *opts=None*)

Run storm code in a schedcoro and log the output messages.

Parameters

- **text** (*str*) – Storm query to execute.
- **opts** – Storm opts.

Returns

None

synapse.lib.module module

class synapse.lib.module.**CoreModule**(*core*, *conf=None*)

Bases: object

confdefs = ()

getConfPath()

Get the path to the module specific config file (conf.yaml).

Notes

This creates the parent directory for the conf.yaml file if it does not exist. This API exists to allow a implementor to get the conf path during initCoreModule and drop a example config if needed. One use case of that is for missing configuration values, an example config can be written to the file and a exception raised.

Returns
Path to where the conf file is located at.

Return type
str

getModDir()

Get the path to the module specific directory.

Notes

This creates the directory if it did not previously exist.

Returns
The filepath to the module specific directory.

Return type
str

getModName()

Return the lowercased name of this module.

Notes

This pulls the `mod_name` attribute on the class. This allows an implementer to set a arbitrary name for the module. If this attribute is not set, it defaults to `self.__class__.__name__.lower()` and sets `mod_name` to that value.

Returns
The module name.

Return type
(str)

getModPath(**paths*)

Construct a path relative to this module's working directory.

Parameters
***paths** – A list of path strings

Notes

This creates the module specific directory if it does not exist.

Returns
The full path (or None if no cortex dir is configured).

Return type
(str)

`getModelDefs()`

`getStormCmds()`

Module implementers may override this to provide a list of Storm commands which will be loaded into the Cortex.

Returns
A list of Storm Command classes (not instances).

Return type
list

`async initCoreModule()`

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

`mod_name = None`

`async preCoreModule()`

Module implementers may override this method to execute code immediately after a module has been loaded.

Notes

The `initCoreModule` function is preferred for overriding instead of `preCoreModule()`.

No Cortex layer/storage operations will function in preCoreModule.

Any exception raised within this method will halt additional loading of the module.

Returns
None

synapse.lib.modules module

Module which implements the synapse module API/convention.

synapse.lib.msgpack module

class synapse.lib.msgpack.**Unpk**

Bases: `object`

An extension of the msgpack streaming Unpacker which reports sizes.

Notes

String objects are decoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow decoding bad input strings.

feed(*byts*)

Feed bytes to the unpacker and return completed objects.

Parameters

byts (*bytes*) – Bytes to unpack.

Notes

It is intended that this function is called multiple times with bytes from some sort of a stream, as it will unpack and return objects as they are available.

Returns

List of tuples containing the item size and the unpacked item.

Return type

list

synapse.lib.msgpack.**deepcopy**(*item*, *use_list=False*)

Copy a msgpack serializable by packing then unpacking it. For complex primitives, this runs in about 1/3 the time of copy.deepcopy()

synapse.lib.msgpack.**dumpfile**(*item*, *path*)

Dump an object to a file by path.

Parameters

- **item** (*object*) – The object to serialize.
- **path** (*str*) – The file path to save.

Returns

None

synapse.lib.msgpack.**en**(*item*)

Use msgpack to serialize a compatible python object.

Parameters

item (*obj*) – The object to serialize

Notes

String objects are encoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow encoding bad input strings.

Returns
The serialized bytes in msgpack format.

Return type
bytes

synapse.lib.msgpack.**getvars**(*varz*)

synapse.lib.msgpack.**isok**(*item*)
Returns True if the item can be msgpacked (by testing packing).

synapse.lib.msgpack.**iterfd**(*fd*)
Generator which unpacks a file object of msgpacked content.

Parameters
fd – File object to consume data from.

Notes

String objects are decoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow decoding bad input strings.

Yields
Objects from a msgpack stream.

synapse.lib.msgpack.**iterfile**(*path*, *since=-1*)
Generator which yields msgpack objects from a file path.

Parameters
path – File path to open and consume data from.

Notes

String objects are decoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow decoding bad input strings.

Yields
Objects from a msgpack stream.

synapse.lib.msgpack.**loadfile**(*path*)
Load and upack the msgpack bytes from a file by path.

Parameters
path (*str*) – The file path to a message pack file.

Raises
msgpack.exceptions.ExtraData – If the file contains multiple objects.

Returns
The decoded python object.

Return type
(obj)

synapse.lib.msgpack.**un**(*byts*, *use_list=False*)

Use msgpack to de-serialize a python object.

Parameters
byts (*bytes*) – The bytes to de-serialize

Notes

String objects are decoded using utf8 encoding. In order to handle potentially malformed input, `unicode_errors='surrogatepass'` is set to allow decoding bad input strings.

Returns
The de-serialized object

Return type
obj

synapse.lib.multislabseqn module

class synapse.lib.multislabseqn.**MultiSlabSeqn**

Bases: *Base*

An append-optimized sequence of byte blobs stored across multiple slabs for fast rotating/culling

async **add**(*item: Any*, *indx=None*) → int

Add a single item to the sequence.

async **cull**(*offs: int*) → bool

Remove entries up to (and including) the given offset.

async **get**(*offs: int*) → Any

Retrieve a single row by offset

getOffsetEvent(*offs: int*) → Event

Returns an asyncio Event that will be set when the particular offset is written. The event will be set if the offset has already been reached.

async **gets**(*offs*, *wait=True*) → AsyncIterator[Tuple[int, Any]]

Just like iter, but optionally waits for new entries once the end is reached.

index() → int

Return the current index to be used

async **iter**(*offs: int*) → AsyncIterator[Tuple[int, Any]]

Iterate over items in a sequence from a given offset.

Parameters
offs (*int*) – The offset to begin iterating from.

Yields
(indx, valu) – The index and valu of the item.

async **last**() → Tuple[int, Any] | None

async rotate() → int

Rotate the Nexus log at the current index.

Note: After this executes the tailseqn will be empty. Waiting for this indx to be written will indicate when it is possible to cull 1 minus the return value such that the rotated seqn is deleted.

Returns
The starting index of the new seqn

Return type
int

setIndex(*indx: int*) → None

static slabFilename(*dirn: str*, *indx: int*)

async waitForOffset(*offs: int*, *timeout=None*) → bool

Returns
true if the event got set, False if timed out

synapse.lib.nexus module

class synapse.lib.nexus.ChangeDist

Bases: *Base*

A utility class to distribute new change entries to mirrors/followers

update() → bool

class synapse.lib.nexus.NexsRoot

Bases: *Base*

async addWriteHold(*reason*)

async cull(*offs*)

async delWriteHold(*reason*)

async eat(*nexsiden*, *event*, *args*, *kwargs*, *meta*)

Actually mutate for the given nexsiden instance.

async enNexsLog()

getChangeDist(*offs: int*) → AsyncIterator[*ChangeDist*]

async index()

async isNexsReady()

async issue(*nexsiden*, *event*, *args*, *kwargs*, *meta=None*)

If I'm not a follower, mutate, otherwise, ask the leader to make the change and wait for the follower loop to hand me the result through a future.

async iter(*offs: int*, *tellready=False*) → AsyncIterator[Any]

Returns an iterator of change entries in the log async **promote**()

async **recover**() → None

Replays the last entry in the nexus log in case we crashed between writing the log and applying it.

Notes

This must be called at cell startup after subsystems are initialized but before any write transactions might happen.

The log can only have recorded 1 entry ahead of what is applied. All log actions are idempotent, so replaying the last action that (might have) already happened is harmless.

reqNotReadOnly()

async **rotate**()

async **runMirrorLoop**(*proxy*)

async **setNexsReady**(*status*)

async **setindex**(*indx*)

async **startup**()

async **waitOffs**(*offs*, *timeout=None*)

class synapse.lib.nexus.**Pusher**

Bases: *Base*

A mixin-class to manage distributing changes where one might plug in mirroring or consensus protocols

classmethod **onPush**(*event: str*, *passitem=False*) → Callable

Decorator that registers a method to be a handler for a named event

Parameters

- **event** – string that distinguishes one handler from another. Must be unique per Pusher subclass
- **passitem** – whether to pass the (offs, mesg) tuple to the handler as "nexsitem"

classmethod **onPushAuto**(*event: str*, *passitem=False*) → Callable

Decorator that does the same as onPush, except automatically creates the top half method

Parameters

- **event** – string that distinguishes one handler from another. Must be unique per Pusher subclass
- **passitem** – whether to pass the (offs, mesg) tuple to the handler as "nexsitem"

async **saveToNexs**(*name*, **args*, ***kwargs*)

setNexsRoot(*nexsroot*)

class synapse.lib.nexus.**RegMethType**(*name: str*, *bases: List[type]*, *attrs: Dict[str, Any]*)

Bases: type

Metaclass that collects all methods in class with _regme prop into a class member called _regclstupls

synapse.lib.node module

class synapse.lib.node.**Node**(*snap*, *sode*, *bylayer=None*)

Bases: `object`

A Cortex hypergraph node.

NOTE: This object is for local Cortex use during a single Xact.

async **addEdge**(*verb*, *n2iden*)

async **addTag**(*tag*, *valu=(None, None)*)

Add a tag to a node.

Parameters

- **tag** (*str*) – The tag to add to the node.
- **valu** – The optional tag value. If specified, this must be a value that norms as a valid time interval as an ival.

Returns

This returns None.

Return type

None

async **delEdge**(*verb*, *n2iden*)

async **delTag**(*tag*, *init=False*)

Delete a tag from the node.

async **delTagProp**(*tag*, *name*)

async **delete**(*force=False*)

Delete a node from the cortex.

The following tear-down operations occur in order:

- validate that you have permissions to delete the node
- validate that you have permissions to delete all tags
- validate that there are no remaining references to the node.
- **delete all the tags (bottom up)**
  - fire onDelTag() handlers
  - delete tag properties from storage
- **delete all secondary properties**
  - fire onDelProp handler
  - delete secondary property from storage
- **delete the primary property**
  - fire onDel handlers for the node
  - delete primary property from storage

async **filter**(*runt*, *text*, *opts=None*, *path=None*) `get`(*name*)

Return a secondary property value from the Node.

Parameters
`name` (*str*) – The name of a secondary property.

Returns
The secondary property value or None.

Return type
(obj)

`getByLayer()`

Return a dictionary that translates the node's bylayer dict to a primitive.

`async getData`(*name*, *defv=None*)

`async getEmbeds`(*embeds*)

Return a dictionary of property embeddings.

`getNodeRefs()`

Return a list of (prop, (form, valu)) refs out for the node.

`async getStorNodes()`

Return a list of the raw storage nodes for each layer.

`getTag`(*name*, *defval=None*)

`getTagProp`(*tag*, *prop*, *defval=None*)

Return the value (or defval) of the given tag property.

`getTagProps`(*tag*)

`getTags`(*leaf=False*)

`has`(*name*)

`async hasData`(*name*)

`hasTag`(*name*)

`hasTagProp`(*tag*, *prop*)

Check if a #foo.bar:baz tag property exists on the node.

`iden()`

`async iterData()`

`async iterDataKeys()`

`async iterEdgeVerbs`(*n2buid*)

`async iterEdgesN1`(*verb=None*)

`async iterEdgesN2`(*verb=None*)

`pack`(*dorepr=False*)

Return the serializable/packed version of the node.

Parameters
`dorepr` (*bool*) – Include repr information for human readable versions of properties.

Returns
An (ndef, info) node tuple.

Return type
(tuple)

async **pop**(*name*, *init=False*)

Remove a property from a node and return the value

async **popData**(*name*)

repr(*name=None*, *defv=None*)

reprs()

Return a dictionary of repr values for props whose repr is different than the system mode value.

async **set**(*name*, *valu*, *init=False*)

Set a property on the node.

Parameters

- **name** (*str*) – The name of the property.
- **valu** (*obj*) – The value of the property.
- **init** (*bool*) – Set to True to disable read-only enforcement

Returns
True if the property was changed.

Return type
(bool)

async **setData**(*name*, *valu*)

async **setTagProp**(*tag*, *name*, *valu*)

Set the value of the given tag property.

async **storm**(*runt*, *text*, *opts=None*, *path=None*)

Parameters
path (Path) – If set, then vars from path are copied into the new runtime, and vars are copied back out into path at the end

Note: If opts is not None and opts['vars'] is set and path is not None, then values of path vars take precedent

tagpropreprs()

Return a dictionary of repr values for tagprops whose repr is different than the system mode value.

class synapse.lib.node.**Path**(*vars*, *nodes*)

Bases: object

A path context tracked through the storm runtime.

clone()

finiframe()

Pop a scope frame from the path, restoring runt if at the top :param runt: A storm runtime to restore if we're at the top :type runt: Runtime :param merge: Set to true to merge vars back up into the next frame :type merge: bool

fork(*node*)

getVar(*name*, *defv=<synapse.common.NoValu object>*)

initframe(*initvars=None*)

meta(*name*, *valu*)

Add node specific metadata to be returned with the node.

async **pack**(*path=False*)

async **popVar**(*name*)

async **setVar**(*name*, *valu*)

synapse.lib.node.**iden**(*pode*)

Return the iden (buid) of the packed node.

Parameters

pode (*tuple*) – A packed node.

Returns

The node iden.

Return type

str

synapse.lib.node.**ndef**(*pode*)

Return a node definition (<form>,<valu>) tuple from the node.

Parameters

pode (*tuple*) – A packed node.

Returns

The (<form>,<valu>) tuple for the node

Return type

((str,obj))

synapse.lib.node.**prop**(*pode*, *prop*)

Return the valu of a given property on the node.

Parameters

- **pode** (*tuple*) – A packed node.
- **prop** (*str*) – Property to retrieve.

Notes

The prop argument may be the full property name (foo:bar:baz), relative property name (:baz) , or the unadorned property name (baz).

Returns:

synapse.lib.node.**props**(*pode*)

Get the props from the node.

Parameters

pode (*tuple*) – A packed node.

Notes

This will include any universal props present on the node.

Returns
A dictionary of properties.

Return type
dict

synapse.lib.node.**reprNdef**(*pode*)

Get the ndef of the pode with a human readable value.

Parameters
pode (*tuple*) – A packed node.

Notes

The human readable value is only available if the node came from a storm query execution where the `repr` key was passed into the `opts` argument with a True value.

Returns
A tuple of form and the human readable value.

Return type
(str, str)

synapse.lib.node.**reprProp**(*pode*, *prop*)

Get the human readable value for a secondary property from the pode.

Parameters

- **pode** (*tuple*) – A packed node.
- **prop** –

Notes

The human readable value is only available if the node came from a storm query execution where the `repr` key was passed into the `opts` argument with a True value.

The prop argument may be the full property name (foo:bar:baz), relative property name (:baz) , or the unadorned property name (baz).

Returns
The human readable property value. If the property is not present, returns None.

Return type
str

synapse.lib.node.**reprTag**(*pode*, *tag*)

Get the human readable value for the tag timestamp from the pode.

Parameters

- **pode** (*tuple*) – A packed node.
- **tag** (*str*) – The tag to get the value for.

Notes

The human readable value is only available if the node came from a storm query execution where the `repr` key was passed into the `opts` argument with a True value.

If the tag does not have a timestamp, this returns a empty string.

Returns
The human readable value for the tag. If the tag is not present, returns None.

Return type
str

`synapse.lib.node.reprTagProps(pode, tag)`

Get the human readable values for any tagprops on a tag for a given node.

Parameters

- **pode** (*tuple*) – A packed node.
- **tag** (*str*) – The tag to get the tagprops reprs for.

Notes

The human readable value is only available if the node came from a storm query execution where the `repr` key was passed into the `opts` argument with a True value.

If the tag does not have any tagprops associated with it, this returns an empty list.

Returns
A list of tuples, containing the name of the tagprop and the repr value.

Return type
list

`synapse.lib.node.tagged(pode, tag)`

Check if a packed node has a given tag.

Parameters

- **pode** (*tuple*) – A packed node.
- **tag** (*str*) – The tag to check.

Examples

Check if a node is tagged with "woot" and dostuff if it is.

if s_node.tagged(node,'woot'):
dostuff()

Notes

If the tag starts with #, this is removed prior to checking.

Returns
True if the tag is present. False otherwise.

Return type
bool

`synapse.lib.node.tags`(*pode*, *leaf=False*)

Get all the tags for a given node.

Parameters

- **pode** (*tuple*) – A packed node.
- **leaf** (*bool*) – If True, only return leaf tags

Returns
A list of tag strings.

Return type
list

`synapse.lib.node.tagsnice`(*pode*)

Get all the leaf tags and the tags that have values or tagprops.

Parameters
pode (*tuple*) – A packed node.

Returns
A list of tag strings.

Return type
list

synapse.lib.oauth module

class `synapse.lib.oauth.OAuthMixin`

Bases: *Pusher*

Mixin for Cells to organize and execute OAuth token refreshes.

async `addOAuthProvider`(*conf*)

async `clearOAuthAccessToken`(*provideriden*, *useriden*)

Remove a client access token by clearing the configuration. This will prevent further refreshes (if scheduled), and a new auth code will be required the next time an access token is requested.

async `delOAuthProvider`(*iden*)

async `getOAuthAccessToken`(*provideriden*, *useriden*)

async `getOAuthClient`(*provideriden*, *useriden*)

async `getOAuthProvider`(*iden*)

listOAuthClients()

Returns
List of (provideriden, useriden, conf) for each client.

Return type
list

async **listOAuthProviders**()

async **setOAuthAuthCode**(*provideriden*, *useriden*, *authcode*, *code_verifier=None*)

Typically set as the end result of a successful OAuth flow. An initial access token and refresh token will be immediately requested, and the client will be loaded into the schedule to be background refreshed.

synapse.lib.oauth.**normOAuthTokenData**(*issued_at*, *data*)

Normalize timestamps to be in epoch millis and set expires_at/refresh_at.

synapse.lib.output module

Tools for easily hookable output from cli-like tools.

class synapse.lib.output.**OutPut**

Bases: object

printf(*mesg*, *addnl=True*)

class synapse.lib.output.**OutPutBytes**

Bases: *OutPutFd*

class synapse.lib.output.**OutPutFd**(*fd*, *enc='utf8'*)

Bases: *OutPut*

class synapse.lib.output.**OutPutStr**

Bases: *OutPut*

synapse.lib.parser module

class synapse.lib.parser.**AstConverter**(*text*)

Bases: Transformer

Convert AST from parser into synapse AST, depth first.

If a method with a name that matches the current rule exists, that will be called, otherwise __default__ will be used

cmdrargs(*meta*, *kids*)

embedquery(*meta*, *kids*)

evalvalu(*meta*, *kids*)

exprdict(*meta*, *kids*)

exprlist(*meta*, *kids*)

funcargs(*meta*, *kids*)

A list of function parameters (as part of a function definition)

funccall(*meta*, *kids*)

liftreverse(*meta*, *kids*)

metaToAstInfo(*meta*, *isterm=False*)

operrelprop_join(*meta*, *kids*)

operrelprop_pivot(*meta*, *kids*, *isjoin=False*)

raiseBadSyntax(*mesg*, *astinfo*)

stormcmdargs(*meta*, *kids*)

subquery(*meta*, *kids*)

switchcase(*meta*, *kids*)

varderef(*meta*, *kids*)

varlist(*meta*, *kids*)

yieldvalu(*meta*, *kids*)

class synapse.lib.parser.**AstInfo**(*text*, *soff*, *eoff*, *sline*, *eline*, *scol*, *ecol*, *isterm*)

Bases: `tuple`

ecol

Alias for field number 6

eline

Alias for field number 4

eoff

Alias for field number 2

isterm

Alias for field number 7

scol

Alias for field number 5

sline

Alias for field number 3

soff

Alias for field number 1

text

Alias for field number 0

class synapse.lib.parser.**CmdStringer**(*visit_tokens: bool = True*)

Bases: `Transformer`

alist(*meta*, *kids*)

cmdstring(*meta*, *kids*)

valu(*meta*, *kids*)

class synapse.lib.parser.**Parser**(*text*, *offs=0*)

Bases: object

Storm query parser

cmdrargs()

Parse command args that might have storm queries as arguments

eval()

lookup()

query()

Parse the storm query

Returns (s_ast.Query): instance of parsed query

search()

synapse.lib.parser.**format_unescape**(*valu*)

synapse.lib.parser.**massage_vartokn**(*astinfo*, *x*)

synapse.lib.parser.**parseEval**(*text*)

synapse.lib.parser.**parseQuery**(*text*, *mode='storm'*)

Parse a storm query and return the Lark AST. Cached here to speed up unit tests

synapse.lib.parser.**parse_cmd_string**(*text*, *off*)

Parse a command line string which may be quoted.

synapse.lib.parser.**unescape**(*valu*)

Parse a string for backslash-escaped characters and omit them. The full list of escaped characters can be found at https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals

synapse.lib.queue module

class synapse.lib.queue.**AQueue**

Bases: *Base*

An async queue with chunk optimized sync compatible consumer.

put(*item*)

Add an item to the queue.

async slice()

class synapse.lib.queue.**Queue**(*maxsize=None*)

Bases: object

An asyncio Queue with batch methods and graceful close.

async close()

async put(*item*)

async puts(*items*)

async size()

async slice(*size=1000*)

async slices(*size=1000*)

class synapse.lib.queue.**Window**

Bases: *Base*

A Queue like object which yields added items. If the queue ever reaches its maxsize, it will be fini()d. On fini(), the Window will continue to yield results until empty and then return.

async **put**(*item*)

Add a single item to the Window.

async **puts**(*items*)

Add multiple items to the window.

synapse.lib.ratelimit module

class synapse.lib.ratelimit.**RateLimit**(*rate*, *per*)

Bases: object

A RateLimit class may be used to detect/enforce rate limits.

Example

allow 20 uses per 10 sec (2/sec) rlimit = RateLimit(20,10)

Notes

It is best (even in a "calls per day" type config) to specify a smaller "per" to force rate "smoothing".

allows()

Returns True if the rate limit has not been reached.

Example

if not rlimit.allows():
rasie RateExceeded()

ok to go…

synapse.lib.reflect module

synapse.lib.reflect.**getClsNames**(*item*)

Return a list of "fully qualified" class names for an instance.

Example

for name in getClsNames(foo):
print(name)

synapse.lib.reflect.**getItemLocals**(*item*)

Iterate the locals of an item and yield (name,valu) pairs.

Example

for name,valu in getItemLocals(item):
dostuff()

synapse.lib.reflect.**getMethName**(*meth*)

Return a fully qualified string for the <mod>.<class>.<func> name of a given method.

synapse.lib.reflect.**getShareInfo**(*item*)

Get a dictionary of special annotations for a Telepath Proxy.

Parameters
item – Item to inspect.

Notes

This will set the `_syn_telemeth` attribute on the item and the items class, so this data is only computed once.

Returns
A dictionary of methods requiring special handling by the proxy.

Return type
dict

synapse.lib.rstorm module

class synapse.lib.rstorm.**OutPutRst**

Bases: *OutPutStr*

Rst specific helper for output intended to be indented in RST text as a literal block.

prefix = ' '

printf(*mesg*, *addnl=True*)

class synapse.lib.rstorm.**StormCliOutput**

Bases: *StormCli*

async **handleErr**(*mesg*)

printf(*mesg*, *addnl=True*, *color=None*)

async **runRstCmdLine**(*text*, *ctx*, *stormopts=None*)

class synapse.lib.rstorm.**StormOutput**(*core*, *ctx*, *stormopts=None*, *opts=None*)

Bases: *StormCmd*

Produce standard output from a stream of storm runtime messages. Must be instantiated for a single query with a rstorm context.

printf(*mesg*, *addnl=True*, *color=None*)

async runCmdLine(*line*)

Run a line of command input for this command.

Parameters

line (*str*) – Line to execute

Examples

Run the foo command with some arguments:

await foo.runCmdLine('foo –opt baz woot.com')

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters

opts (*dict*) – Options dictionary.

class synapse.lib.rstorm.**StormRst**

Bases: *Base*

async run()

Parses the specified RST file with Storm directive handling.

Returns

List of line strings for the RST output

Return type

list

synapse.lib.rstorm.**getCell**(*ctor*, *conf*)

synapse.lib.schemas module

synapse.lib.scope module

class synapse.lib.scope.**Scope**(**frames*, ***vals*)

Bases: object

The Scope object assists in creating nested varible scopes.

Example

with Scope() as scope:

scope.set('foo',10)

with scope:
scope.set('foo',20) dostuff(scope) # 'foo' is 20…

dostuff(scope) # 'foo' is 10 again…

add(*name*, **vals*)

Add values as iter() compatible items in the current scope frame.

copy()

Create a shallow copy of the current Scope.

Returns
A new scope which is a copy of the current scope.

Return type
Scope

enter(*vals=None*)

Add an additional scope frame.

get(*name*, *defval=None*)

Retrieve a value from the closest scope frame.

iter(*name*)

Iterate through values added with add() from each scope frame.

leave()

Pop the current scope frame.

pop(*name*, *defval=None*)

Pop and return a value (from the last frame) of the scope.

Parameters
name (*str*) – The name of the scope variable.

Returns
The scope variable value or None

Return type
obj

set(*name*, *valu*)

Set a value in the current scope frame.

update(*vals*)

Set multiple values in the current scope frame.

synapse.lib.scope.**clone**(*task: Task*) → None

Clone the current task Scope onto the provided task.

Parameters
task (*asyncio.Task*) – The task object to attach the scope too.

Notes

This must be run from an asyncio IO loop.

If the current task does not have a scope, we clone the default global Scope.

This will `enter()` the scope, and add a task callback to `leave()` the scope.

Returns
None

`synapse.lib.scope.ctor`(*name*, *func*, **args*, ***kwargs*)

Add a ctor callback to the global scope.

`synapse.lib.scope.enter`(*vals=None*)

Return the task's local scope for use in a with block

`synapse.lib.scope.get`(*name*, *defval=None*)

Access this task's scope with default values from glob.

`synapse.lib.scope.pop`(*name*)

Pop and return a task scope variable. :param name: The task scope variable name. :type name: str

Returns
The scope value or None

Return type
obj

`synapse.lib.scope.set`(*name*, *valu*)

Set a value in the current frame of the local task scope.

`synapse.lib.scope.update`(*vals*)

synapse.lib.scrape module

`synapse.lib.scrape.contextScrape`(*text*, *form=None*, *refang=True*, *first=False*)

Scrape types from a blob of text and yield info dictionaries.

Parameters

- **text** (*str*) – Text to scrape.
- **form** (*str*) – Optional form to scrape. If present, only scrape items which match the provided form.
- **refang** (*bool*) – Whether to remove de-fanging schemes from text before scraping.
- **first** (*bool*) – If true, only yield the first item scraped.

Notes

The dictionaries yielded by this function contains the following keys:

match
The raw matching text found in the input text.

offset
The offset into the text where the match was found.

valu
The resulting value.

form
The corresponding form for the valu.

Returns
Yield info dicts of results.

Return type
(dict)

async synapse.lib.scrape.**contextScrapeAsync**(*text*, *form=None*, *refang=True*, *first=False*)

Scrape types from a blob of text and yield info dictionaries, potentially in a spawned process.

Parameters

- **text** (*str*) – Text to scrape.
- **form** (*str*) – Optional form to scrape. If present, only scrape items which match the provided form.
- **refang** (*bool*) – Whether to remove de-fanging schemes from text before scraping.
- **first** (*bool*) – If true, only yield the first item scraped.

Notes

The dictionaries yielded by this function contains the following keys:

match
The raw matching text found in the input text.

offset
The offset into the text where the match was found.

valu
The resulting value.

form
The corresponding form for the valu.

Returns
Yield info dicts of results.

Return type
(dict)

synapse.lib.scrape.**cve_check**(*match: Match*) synapse.lib.scrape.**fqdn_check**(*match: Match*)

synapse.lib.scrape.**fqdn_prefix_check**(*match: Match*)

synapse.lib.scrape.**genFangRegex**(*fangs*, *flags=RegexFlag.I*)

synapse.lib.scrape.**genMatches**(*text: str*, *regx: compile*, *opts: dict*)

Generate regular expression matches for a blob of text.

Parameters

- **text** (*str*) – The text to generate matches for.
- **regx** (*regex.Regex*) – A compiled regex object. The regex must contained a named match group for `valu`.
- **opts** (*dict*) – An options dictionary.

Notes

The dictionaries yielded by this function contains the following keys:

raw_valu
The raw matching text found in the input text.

offset
The offset into the text where the match was found.

valu
The resulting value - this may be altered by callbacks.

The options dictionary can contain a `callback` key. This function is expected to take a single argument, a regex.Match object, and return a tuple of the new valu and info dictionary. The new valu is used as the `valu` key in the returned dictionary, and any other information in the info dictionary is pushed into the return dictionary as well.

Yields
dict – A dictionary of match results.

async synapse.lib.scrape.**genMatchesAsync**(*text: str*, *regx: compile*, *opts: dict*)

Generate regular expression matches for a blob of text, potentially in a spawned process.

Parameters

- **text** (*str*) – The text to generate matches for.
- **regx** (*regex.Regex*) – A compiled regex object. The regex must contained a named match group for `valu`.
- **opts** (*dict*) – An options dictionary.

Notes

The dictionaries yielded by this function contains the following keys:

raw_valu
: The raw matching text found in the input text.

offset
: The offset into the text where the match was found.

valu
: The resulting value - this may be altered by callbacks.

The options dictionary can contain a `callback` key. This function is expected to take a single argument, a regex.Match object, and return a tuple of the new valu and info dictionary. The new valu is used as the `valu` key in the returned dictionary, and any other information in the info dictionary is pushed into the return dictionary as well.

Yields
: *dict* – A dictionary of match results.

`synapse.lib.scrape.`**`getForms`**`()`

Get a list of forms recognized by the scrape APIs.

Returns
: A list of form values.

Return type
: list

`synapse.lib.scrape.`**`inet_server_check`**`(`*match: Match*`)`

`synapse.lib.scrape.`**`ipv6_check`**`(`*match: Match*`)`

`synapse.lib.scrape.`**`linux_path_check`**`(`*match: Match*`)`

`synapse.lib.scrape.`**`refang_text`**`(`*txt*`)`

Remove address de-fanging in text blobs, .e.g. example[.]com to example.com

Matches to keys in FANGS is case-insensitive, but replacement will always be with the lowercase version of the re-fanged value. For example, HXXP://FOO.COM will be returned as http://FOO.COM

Returns
: Re-fanged text blob

Return type
: (str)

`synapse.lib.scrape.`**`refang_text2`**`(`*txt: str, re: compile = regex.Regex('fxp:|fxps:|hxxp:|hxxps:|fxp\\[s\\]:|hxxp\\[s\\]:|ftp\\[:\\]|fxp\\[:\\]|ftps\\[:\\]|fxps\\[flags=regex.I | regex.V0), fangs: dict = {'(.)': '.', '()': '', '()': '', '()': '', '.]': '.', '[.': '.', '[.]': '.', '[:]': ':', '[@]': '@', '[at]': '@', '[dot]': '.', '[]': '', '[]': '', '[]': '', '\\.': '.', 'ftp(:)': 'ftp:', 'ftp[:': 'ftp:', 'ftp[://]': 'ftp://', 'ftp[:]': 'ftp:', 'ftps(:)': 'ftps:', 'ftps[:': 'ftps:', 'ftps[://]': 'ftps://', 'ftps[:]': 'ftps:', 'fxp(:)': 'ftp:', 'fxp:': 'ftp:', 'fxp[:': 'ftp:', 'fxp[://]': 'ftp://', 'fxp[:]': 'ftp:', 'fxp[s]:': 'ftps:', 'fxps(:)': 'ftps:', 'fxps:': 'ftps:', 'fxps[:': 'ftps:', 'fxps[://]': 'ftps://', 'fxps[:]': 'ftps:', 'http(:)': 'http:', 'http[:': 'http:', 'http[://]': 'http://', 'http[:]': 'http:', 'https(:)': 'https:', 'https[:': 'https:', 'https[://]': 'https://', 'https[:]': 'https:', 'hxxp(:)': 'http:', 'hxxp:': 'http:', 'hxxp[:': 'http:', 'hxxp[://]': 'http://', 'hxxp[:]': 'http:', 'hxxp[s]:': 'https:', 'hxxps(:)': 'https:', 'hxxps:': 'https:', 'hxxps[:': 'https:', 'hxxps[://]': 'https://', 'hxxps[:]': 'https:'}*`)`

Remove address de-fanging in text blobs, .e.g. example[.]com to example.com

Notes

Matches to keys in FANGS is case-insensitive, but replacement will always be with the lowercase version of the re-fanged value. For example, `HXXP://FOO.COM` will be returned as `http://FOO.COM`

Parameters
txt (*str*) – The text to re-fang.

Returns
A tuple containing the new text, and a dictionary containing offset information where the new text was altered with respect to the original text.

Return type
tuple(str, dict)

`synapse.lib.scrape.`**`scrape`**(*text*, *ptype=None*, *refang=True*, *first=False*)

Scrape types from a blob of text and return node tuples.

Parameters

- **text** (*str*) – Text to scrape.
- **ptype** (*str*) – Optional ptype to scrape. If present, only scrape items which match the provided type.
- **refang** (*bool*) – Whether to remove de-fanging schemes from text before scraping.
- **first** (*bool*) – If true, only yield the first item scraped.

Returns
Yield tuples of node ndef values.

Return type
(str, object)

`async` `synapse.lib.scrape.`**`scrapeAsync`**(*text*, *ptype=None*, *refang=True*, *first=False*)

Scrape types from a blob of text and return node tuples, potentially in a spawned process.

Parameters

- **text** (*str*) – Text to scrape.
- **ptype** (*str*) – Optional ptype to scrape. If present, only scrape items which match the provided type.
- **refang** (*bool*) – Whether to remove de-fanging schemes from text before scraping.
- **first** (*bool*) – If true, only yield the first item scraped.

Returns
Yield tuples of node ndef values.

Return type
(str, object)

`synapse.lib.scrape.`**`unc_path_check`**(*match: Match*)

`synapse.lib.scrape.`**`windows_path_check`**(*match: Match*)

synapse.lib.share module

class synapse.lib.share.**Share**

Bases: *Base*

Class to wrap a dynamically shared object.

synapse.lib.slaboffs module

class synapse.lib.slaboffs.**SlabOffs**(*slab: Slab*, *db: str*)

Bases: object

A helper for storing offset integers by iden.

As with all slab objects, this is meant for single-thread async loop use.

delete(*iden*)

get(*iden*)

set(*iden*, *offs*)

synapse.lib.slabseqn module

class synapse.lib.slabseqn.**SlabSeqn**(*slab*, *name: str*)

Bases: object

An append optimized sequence of byte blobs.

Parameters

- **lenv** (*lmdb.Environment*) – The LMDB Environment.
- **name** (*str*) – The name of the sequence.

add(*item*, *indx=None*)

Add a single item to the sequence.

async aiter(*offs*, *wait=False*, *timeout=None*)

Iterate over items in a sequence from a given offset.

Parameters

- **offs** (*int*) – The offset to begin iterating from.
- **wait** (*boolean*) – Once caught up, yield new results in realtime.
- **timeout** (*int*) – Max time to wait for a new item.

Yields

(indx, valu) – The index and valu of the item.

async cull(*offs*)

Remove entries up to (and including) the given offset.

first()

get(*offs*)

Retrieve a single row by offset

`getByIndxByts`(*indxbyts*)

`getOffsetEvent`(*offs*)

Returns an asyncio Event that will be set when the particular offset is written. The event will be set if the offset has already been reached.

`getraw`(*byts*)

async `gets`(*offs*, *wait=True*)

Returns an async generator of indx/valu tuples, optionally waiting and continuing to yield them as new entries are added

Parameters

- **offs** (*int*) – The offset to begin iterating from.
- **wait** (*bool*) – Whether to continue yielding tupls when it hits the end of the sequence.

Yields

(indx, valu) – The index and valu of the item.

`index`()

Return the current index to be used

`iter`(*offs*)

Iterate over items in a sequence from a given offset.

Parameters

offs (*int*) – The offset to begin iterating from.

Yields

(indx, valu) – The index and valu of the item.

`iterBack`(*offs*)

Iterate backwards over items in a sequence from a given offset.

Parameters

offs (*int*) – The offset to begin iterating from.

Yields

(indx, valu) – The index and valu of the item.

`last`()

`nextindx`()

Determine the next insert offset according to storage.

Returns

The next insert offset.

Return type

int

`pop`(*offs*)

Pop a single entry at the given offset.

`rows`(*offs*)

Iterate over raw indx, bytes tuples from a given offset.

`save(items)`

Save a series of items to a sequence.

Parameters
`items` (*tuple*) – The series of items to save into the sequence.

Returns
The index of the first item

`slice(offs, size)`

`sliceBack(offs, size)`

`stat()`

`trim(offs)`

Delete entries starting at offset and moving forward.

`async waitForOffset(offs, timeout=None)`

Returns
true if the event got set, False if timed out

synapse.lib.snap module

`class synapse.lib.snap.ProtoNode(ctx, buid, form, valu, node)`

Bases: `object`

A prototype node used for staging node adds using a SnapEditor.

TODO: This could eventually fully mirror the synapse.lib.node.Node API and be used
to slipstream into sections of the pipeline to facilitate a bulk edit / transaction

`async addEdge(verb, n2iden)`

`async addTag(tag, valu=(None, None), tagnode=None)`

`async delEdge(verb, n2iden)`

`get(name)`

`async getData(name)`

`getNodeEdit()`

`async getSetOps(name, valu, norminfo=None)`

`getTag(tag)`

`getTagProp(tag, name)`

`iden()`

`async set(name, valu, norminfo=None)`

`async setData(name, valu)`

`async setTagProp(tag, name, valu)`

class synapse.lib.snap.**Scrubber**(*rules*)

Bases: object

scrub(*pode*)

class synapse.lib.snap.**Snap**

Bases: *Base*

A "snapshot" is a transaction across multiple Cortex layers.

The Snap object contains the bulk of the Cortex API to facilitate performance through careful use of transaction boundaries.

Transactions produce the following EventBus events:

('print', {}),

async **addFeedData**(*name*, *items*)

async **addFeedNodes**(*name*, *items*)

Call a feed function and return what it returns (typically yields Node()s).

Parameters

- **name** (*str*) – The name of the feed record type.
- **items** (*list*) – A list of records of the given feed type.

Returns

The return value from the feed function. Typically Node() generator.

Return type

(object)

async **addNode**(*name*, *valu*, *props=None*, *norminfo=None*)

Add a node by form name and value with optional props.

Parameters

- **name** (*str*) – The form of node to add.
- **valu** (*obj*) – The value for the node.
- **props** (*dict*) – Optional secondary properties for the node.

Notes

If a props dictionary is provided, it may be mutated during node construction.

Returns

A Node object. It may return None if the snap is unable to add or lift the node.

Return type

s_node.Node

async **addNodes**(*nodedefs*)

Add/merge nodes in bulk.

The addNodes API is designed for bulk adds which will also set properties, add tags, add edges, and set nodedata to existing nodes. Nodes are specified as a list of the following tuples:

((form, valu), {'props':{}, 'tags':{}})

Parameters
`nodedefs` (*list*) – A list of nodedef tuples.

Returns
A list of xact messages.

Return type
(list)

async **addStormRuntime**(*query*, *opts=None*, *user=None*)

async **applyNodeEdit**(*edit*, *nodecache=None*)

async **applyNodeEdits**(*edits*, *nodecache=None*)

Sends edits to the write layer and evaluates the consequences (triggers, node object updates)

buidcachesize = 100000

async **clearCache**()

clearCachedNode(*buid*)

async **eval**(*text*, *opts=None*, *user=None*)

Run a storm query and yield Node() objects.

getEditor()

async **getNodeByBuid**(*buid*)

Retrieve a node tuple by binary id.

Parameters
`buid` (*bytes*) – The binary ID for the node.

Returns
The node object or None.

Return type
Optional[s_node.Node]

async **getNodeByNdef**(*ndef*)

Return a single Node by (form,valu) tuple.

Parameters

- **ndef** (*(str,obj)*) – A (form,valu) ndef tuple. valu must be
- **normalized.** –

Returns
The Node or None.

Return type
(*synapse.lib.node.Node*)

async **getNodeData**(*buid*, *name*, *defv=None*)

Get nodedata from closest to write layer, no merging involved

getNodeEditor(*node*)

async **getRuntNodes**(*full*, *valu=None*, *cmpr=None*)

async getSnapMeta()
Retrieve snap metadata to store along side nodeEdits.

getStormRuntime(*query*, *opts=None*, *user=None*)

async getTagNode(*name*)
Retrieve a cached tag node. Requires name is normed. Does not add.

async getTagNorm(*tagname*)

async hasNodeData(*buid*, *name*)
Return True if the buid has nodedata set on it under the given name False otherwise

async hasNodeEdge(*buid1*, *verb*, *buid2*)

async iterEdgeVerbs(*n1buid*, *n2buid*)

async iterNodeData(*buid*)
Returns: Iterable[Tuple[str, Any]]

async iterNodeDataKeys(*buid*)
Yield each data key from the given node by buid.

async iterNodeEdgesN1(*buid*, *verb=None*)

async iterNodeEdgesN2(*buid*, *verb=None*)

async iterStormPodes(*text*, *opts*, *user=None*)
Yield packed node tuples for the given storm query text.

async keepalive(*period*)

async nodes(*text*, *opts=None*, *user=None*)

async nodesByDataName(*name*)

async nodesByProp(*full*, *reverse=False*)

async nodesByPropArray(*full*, *cmpr*, *valu*, *reverse=False*)

async nodesByPropTypeValu(*name*, *valu*, *reverse=False*)

async nodesByPropValu(*full*, *cmpr*, *valu*, *reverse=False*)

async nodesByTag(*tag*, *form=None*, *reverse=False*)

async nodesByTagProp(*form*, *tag*, *name*, *reverse=False*)

async nodesByTagPropValu(*form*, *tag*, *name*, *cmpr*, *valu*, *reverse=False*)

async nodesByTagValu(*tag*, *cmpr*, *valu*, *form=None*, *reverse=False*)

async printf(*mesg*)

async saveNodeEdits(*edits*, *meta*)

async storm(*text*, *opts=None*, *user=None*)
Execute a storm query and yield (Node(), Path()) tuples.

tagcachesize = 1000

async **warn**(*mesg*, *log=True*, ***info*)

async **warnonce**(*mesg*, *log=True*, ***info*)

class synapse.lib.snap.**SnapEditor**(*snap*)

Bases: object

A SnapEditor allows tracking node edits with subs/deps as a transaction.

async **addNode**(*formname*, *valu*, *props=None*, *norminfo=None*)

async **getAddNodeOps**(*formname*, *valu*, *props=None*, *norminfo=None*)

async **getNodeByBuid**(*buid*)

getNodeEdits()

loadNode(*node*)

synapse.lib.spooled module

class synapse.lib.spooled.**Dict**

Bases: *Spooled*

get(*key*, *defv=None*)

has(*key*)

items()

keys()

pop(*key*, *defv=None*)

async **set**(*key*, *val*)

class synapse.lib.spooled.**Set**

Bases: *Spooled*

A minimal set-like implementation that will spool to a slab on large growth.

async **add**(*valu*)

discard(*valu*)

has(*key*)

class synapse.lib.spooled.**Spooled**

Bases: *Base*

A Base class that can be used to implement objects which fallback to lmdb.

These objects are intended to fallback from Python to lmbd slabs, which aligns them together. Under memory pressure, these objects have a better shot of getting paged out.

synapse.lib.storm module

class synapse.lib.storm.**BackgroundCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute a query pipeline as a background task. NOTE: Variables are passed through but nodes are not

async **execStormCmd**(*runt*, *genr*)

Abstract base method

async **execStormTask**(*query*, *opts*)

getArgParser()

name = 'background'

class synapse.lib.storm.**BatchCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Run a query with batched sets of nodes.

The batched query will have the set of inbound nodes available in the variable $nodes.

This command also takes a conditional as an argument. If the conditional evaluates to true, the nodes returned by the batched query will be yielded, if it evaluates to false, the inbound nodes will be yielded after executing the batched query.

NOTE: This command is intended to facilitate use cases such as queries to external
APIs with aggregate node values to reduce quota consumption. As this command interrupts the node stream, it should be used carefully to avoid unintended slowdowns in the pipeline.

Example

// Execute a query with batches of 5 nodes, then yield the inbound nodes batch $lib.false –size 5 { $lib.print($nodes) }

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'batch'

class synapse.lib.storm.**Cmd**(*runt*, *runtsafe*)

Bases: object

A one line description of the command.

Command usage details and long form description.

Example

cmd –help

Notes

Python Cmd implementers may override the `forms` attribute with a dictionary to provide information about Synapse forms which are possible input and output nodes that a Cmd may recognize. A list of (key, form) tuples may also be added to provide information about forms which may have additional nodedata added to them by the Cmd.

Example:

```
{
    'input': (
        'inet:ipv4',
        'tel:mob:telem',
    ),
    'output': (
        'geo:place',
    ),
    'nodedata': (
        ('foodata', 'inet:http:request'),
        ('bardata', 'inet:ipv4'),
    ),
}
```

asroot = False

async execStormCmd(*runt*, *genr*)
    Abstract base method

forms = {}

getArgParser()

classmethod getCmdBrief()

getDescr()

getName()

classmethod getStorNode(*form*)

isReadOnly()

name = 'cmd'

pkgname = ''

readonly = False

async setArgv(*argv*)

svciden = ''

class synapse.lib.storm.**CopyToCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Copy nodes from the current view into another view.

Examples

// Copy all nodes tagged with #cno.mal.redtree to the target view.

#cno.mal.redtree | copyto 33c971ac77943da91392dadd0eec0571

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'copyto'

class synapse.lib.storm.**CountCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Iterate through query results, and print the resulting number of nodes which were lifted. This does not yield the nodes counted, unless the –yield switch is provided.

Example

Count the number of IPV4 nodes with a given ASN. inet:ipv4:asn=20 | count

Count the number of IPV4 nodes with a given ASN and yield them. inet:ipv4:asn=20 | count –yield

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'count'

readonly = True

class synapse.lib.storm.**DelNodeCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Delete nodes produced by the previous query logic.

(no nodes are returned)

Example

inet:fqdn=vertex.link | delnode

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'delnode'

class synapse.lib.storm.**DiffCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Generate a list of nodes with changes in the top layer of the current view.

Examples

// Lift all nodes with any changes

diff

// Lift ou:org nodes that were added in the top layer.

diff –prop ou:org

// Lift inet:ipv4 nodes with the :asn property modified in the top layer.

diff –prop inet:ipv4:asn

// Lift the nodes with the tag #cno.mal.redtree added in the top layer.

diff –tag cno.mal.redtree

`async execStormCmd(runt, genr)`

Abstract base method

`getArgParser()`

`name = 'diff'`

`readonly = True`

`class synapse.lib.storm.DivertCmd(runt, runtsafe)`

Bases: `Cmd`

Either consume a generator or yield it's results based on a conditional.

NOTE: This command is purpose built to facilitate the –yield convention
common to storm commands.

NOTE: The genr argument must not be a function that returns, else it will
be invoked for each inbound node.

Example

divert $cmdopts.yield $fooBarBaz()

`async execStormCmd(runt, genr)`

Abstract base method

`getArgParser()`

`name = 'divert'`

`class synapse.lib.storm.DmonManager`

Bases: `Base`

Manager for StormDmon objects.

`async addDmon(iden, ddef)`

`getDmon(iden)`

`getDmonDef(iden)`

`getDmonDefs()`

getDmonRunlog(*iden*)

async **popDmon**(*iden*)

Remove the dmon and fini it if its exists.

async **start**()

Start all the dmons.

async **stop**()

Stop all the dmons.

class synapse.lib.storm.**EdgesDelCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Bulk delete light edges from input nodes.

Examples

Delete all "foo" light edges from an inet:ipv4 inet:ipv4=1.2.3.4 | edges.del foo

Delete light edges with any verb from a node inet:ipv4=1.2.3.4 | edges.del *

Delete all "foo" light edges to an inet:ipv4 inet:ipv4=1.2.3.4 | edges.del foo –n2

async **delEdges**(*node*, *verb*, *n2=False*)

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'edges.del'

class synapse.lib.storm.**GraphCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Generate a subgraph from the given input nodes and command line options.

Example

Using the graph command:

```
inet:fqdn | graph
            --degrees 2
            --filter { -#nope }
            --pivot { -> meta:seen }
            --form-pivot inet:fqdn {<- * | limit 20}
            --form-pivot inet:fqdn {-> * | limit 20}
            --form-filter inet:fqdn {-inet:fqdn:issuffix=1}
            --form-pivot syn:tag {-> *}
            --form-pivot * {-> #}
```

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'graph'

class synapse.lib.storm.**HelpCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

List available information about Storm and brief descriptions of different items.

Notes

If an item is provided, this can be a string or a function.

Examples

// Get all available commands, libraries, types, and their brief descriptions.

help

// Only get commands which have "model" in the name.

help model

// Get help about the base Storm library

help $lib

// Get detailed help about a specific library or library function

help –verbose $lib.print

// Get detailed help about a named Storm type

help –verbose str

// Get help about a method from a $node object

<inbound $node> help $node.tags

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'help'

class synapse.lib.storm.**IdenCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Lift nodes by iden.

Example

iden b25bc9eec7e159dce879f9ec85fb791f83b505ac55b346fcb64c3c51e98d1175 | count

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'iden'

readonly = True

class synapse.lib.storm.**IntersectCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Yield an intersection of the results of running inbound nodes through a pivot.

Note: This command must consume the entire inbound stream to produce the intersection. This type of stream consuming before yielding results can cause the query to appear laggy in comparison with normal incremental stream operations.

Examples

// Show the it:mitre:attack:technique nodes common to several groups

it:mitre:attack:group*in=(G0006, G0007) | intersect { -> it:mitre:attack:technique }

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'intersect'

class synapse.lib.storm.**LiftByVerb**(*runt*, *runtsafe*)

Bases: *Cmd*

Lift nodes from the current view by an light edge verb.

Examples

Lift all the n1 nodes for the light edge "foo" lift.byverb "foo"

Lift all the n2 nodes for the light edge "foo" lift.byverb –n2 "foo"

Notes

Only a single instance of a node will be yielded from this command when that node is lifted via the light edge membership.

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

async **iterEdgeNodes**(*verb*, *idenset*, *n2=False*)

name = 'lift.byverb'

class synapse.lib.storm.**LimitCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Limit the number of nodes generated by the query in the given position.

Example

inet:ipv4 | limit 10

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'limit'

readonly = True

class synapse.lib.storm.MaxCmd(*runt*, *runtsafe*)

Bases: *Cmd*

Consume nodes and yield only the one node with the highest value for an expression.

Examples

// Yield the file:bytes node with the highest :size property file:bytes#foo.bar | max :size

// Yield the file:bytes node with the highest value for $tick file:bytes#foo.bar +.seen ($tick, $tock) = .seen | max $tick

// Yield the it:dev:str node with the longest length it:dev:str | max $lib.len($node.value())

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'max'

readonly = True

class synapse.lib.storm.MergeCmd(*runt*, *runtsafe*)

Bases: *Cmd*

Merge edits from the incoming nodes down to the next layer.

NOTE: This command requires the current view to be a fork.

NOTE: The arguments for including/excluding tags can accept tag glob
expressions for specifying tags. For more information on tag glob expressions, check the Synapse documentation for $node.globtags().

Examples

// Having tagged a new #cno.mal.redtree subgraph in a forked view…

#cno.mal.redtree | merge –apply

// Print out what the merge command *would* do but dont.

#cno.mal.redtree | merge

// Merge any org nodes with changes in the top layer.

diff | +ou:org | merge –apply

// Merge all tags other than cno.* from ou:org nodes with edits in the // top layer.

diff | +ou:org | merge –only-tags –exclude-tags cno.** –apply

// Merge only tags rep.vt.* and rep.whoxy.* from ou:org nodes with edits // in the top layer.

diff | +ou:org | merge –include-tags rep.vt.* rep.whoxy.* –apply

// Lift only inet:ipv4 nodes with a changed :asn property in top layer // and merge all changes.

diff –prop inet:ipv4:asn | merge –apply

// Lift only nodes with an added #cno.mal.redtree tag in the top layer and merge them.

diff –tag cno.mal.redtree | merge –apply

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'merge'

class synapse.lib.storm.**MinCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Consume nodes and yield only the one node with the lowest value for an expression.

Examples

// Yield the file:bytes node with the lowest :size property file:bytes#foo.bar | min :size

// Yield the file:bytes node with the lowest value for $tick file:bytes#foo.bar +.seen ($tick, $tock) = .seen | min $tick

// Yield the it:dev:str node with the shortest length it:dev:str | min $lib.len($node.value())

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'min'

readonly = True

class synapse.lib.storm.**MoveNodesCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Move storage nodes between layers.

Storage nodes will be removed from the source layers and the resulting storage node in the destination layer will contain the merged values (merged in bottom up layer order by default).

Examples

// Move storage nodes for ou:org nodes to the top layer

ou:org | movenodes –apply

// Print out what the movenodes command *would* do but dont.

ou:org | movenodes

// In a view with many layers, only move storage nodes from the bottom layer // to the top layer.

$layers = $lib.view.get().layers $top = $layers.0.iden $bot = $layers."-1".iden

ou:org | movenodes –srclayers $bot –destlayer $top

// In a view with many layers, move storage nodes to the top layer and // prioritize values from the bottom layer over the other layers.

$layers = $lib.view.get().layers $top = $layers.0.iden $mid = $layers.1.iden $bot = $layers.2.iden

ou:org | movenodes –precedence $bot $top $mid

`async execStormCmd(runt, genr)`

Abstract base method

`getArgParser()`

`name = 'movenodes'`

`class synapse.lib.storm.MoveTagCmd(runt, runtsafe)`

Bases: *Cmd*

Rename an entire tag tree and preserve time intervals.

Example

movetag foo.bar baz.faz.bar

`async execStormCmd(runt, genr)`

Abstract base method

`getArgParser()`

`name = 'movetag'`

`class synapse.lib.storm.OnceCmd(runt, runtsafe)`

Bases: *Cmd*

The once command is used to filter out nodes which have already been processed via the use of a named key. It includes an optional parameter to allow the node to pass the filter again after a given amount of time.

For example, to run an enrichment command on a set of nodes just once:

file:bytes#my.files | once enrich:foo | enrich.foo

The once command filters out any nodes which have previously been through any other use of the "once" command using the same <name> (in this case "enrich:foo").

You may also specify the –asof option to allow nodes to pass the filter after a given amount of time. For example, the following command will allow any given node through every 2 days:

file:bytes#my.files | once enrich:foo –asof "-2 days" | enrich.foo

Use of "–asof now" or any future date or positive relative time offset will always allow the node to pass the filter.

State tracking data for the once command is stored as nodedata which is stored in your view's write layer, making it view-specific. So if you have two views, A and B, and they do not share any layers between them, and you execute this query in view A:

inet:ipv4=8.8.8.8 | once enrich:address | enrich.baz

And then you run it in view B, the node will still pass through the once command to the enrich.baz portion of the query because the tracking data for the once command does not yet exist in view B.

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'once'

class synapse.lib.storm.**ParallelCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute part of a query pipeline in parallel. This can be useful to minimize round-trip delay during enrichments.

Examples

inet:ipv4#foo | parallel { $place = $lib.import(foobar).lookup(:latlong) [:place=$place] }

NOTE: Storm variables set within the parallel query pipelines do not interact.

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'parallel'

async **nextitem**(*inq*)

async **pipeline**(*runt*, *query*, *inq*, *outq*)

readonly = True

class synapse.lib.storm.**Parser**(*prog=None*, *descr=None*, *root=None*)

Bases: object

add_argument(**names*, ***opts*)

help(*mesg=None*)

parse_args(*argv*)

set_inputs(*idefs*)

class synapse.lib.storm.**PureCmd**(*cdef*, *runt*, *runtsafe*)

Bases: *Cmd*

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

getDescr()

getName()

readonly = True

class synapse.lib.storm.**ReIndexCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Use admin privileges to re index/normalize node properties.

NOTE: Currently does nothing but is reserved for future use.

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'reindex'

class synapse.lib.storm.**RunAsCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute a storm query as a specified user.

NOTE: This command requires admin privileges.

Examples

// Create a node as another user. runas someuser { [inet:fqdn=foo.com] }

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'runas'

class synapse.lib.storm.**Runtime**

Bases: *Base*

A Runtime represents the instance of a running query.

The runtime should maintain a firm API boundary using the snap. Parallel query execution requires that the snap be treated as an opaque object which is called through, but not dereferenced.

addInput(*node*)

Add a Node() object as input to the query runtime.

allowed(*perms*, *gateiden=None*, *default=None*)

allowedEasyPerm(*item*, *perm*)

allowedReason(*perms*, *gateiden=None*, *default=None*)

Similar to allowed, but always prefer the default value specified by the caller. Default values are still pulled from permdefs if there is a match there; but still prefer caller default. This results in a ternary response that can be used to know if a rule had a positive/negative or no match. The matching reason metadata is also returned.

cancel()

confirm(*perms*, *gateiden=None*, *default=None*)

Raise AuthDeny if the user doesn't have the permission.

Notes

An elevated runtime with asroot=True will always return True.

Parameters

- **perms** (*tuple*) – The permission tuple.
- **gateiden** (*str*) – The gateiden.
- **default** (*bool*) – The default value.

Returns
If the permission is allowed.

Return type
True

Raises
AuthDeny – If the user does not have the permission.

confirmEasyPerm(*item*, *perm*, *mesg=None*)

confirmPropDel(*prop*, *layriden=None*)

confirmPropSet(*prop*, *layriden=None*)

async **coreDynCall**(*todo*, *perm=None*)

async **dyncall**(*iden*, *todo*, *gatekeys=()*)

async **dyniter**(*iden*, *todo*, *gatekeys=()*)

async **emit**(*item*)

async **emitter**()

async **execute**(*genr=None*)

getCmdRuntime(*query*, *opts=None*)

Yield a runtime with proper scoping for use in executing a pure storm command.

getGraph()

async **getInput**()

async **getModRuntime**(*query*, *opts=None*)

Construct a non-context managed runtime for use in module imports.

async **getOneNode**(*propname*, *valu*, *filt=None*, *cmpr='='*)

Return exactly 1 node by <prop> <cmpr> <valu>

getOpt(*name*, *defval=None*)

getScopeVars()

Return a dict of all the vars within this and all parent scopes.

async getStormQuery(*text*)

getSubRuntime(*query*, *opts=None*)

Yield a runtime with shared scope that will populate changes upward.

async getTeleProxy(*url*, ***opts*)

getVar(*name*, *defv=None*)

initPath(*node*)

async initSubRuntime(*query*, *opts=None*)

Construct and return sub-runtime with a shared scope. (caller must fini)

isAdmin(*gateiden=None*)

isRuntVar(*name*)

layerConfirm(*perms*)

async popVar(*name*)

async printf(*mesg*)

reqAdmin(*gateiden=None*, *mesg=None*)

async reqGateKeys(*gatekeys*)

async reqUserCanReadLayer(*layriden*)

setGraph(*gdef*)

setOpt(*name*, *valu*)

async setVar(*name*, *valu*)

async storm(*text*, *opts=None*, *genr=None*)

Execute a storm runtime which inherits from this storm runtime.

tick()

async warn(*mesg*, ***info*)

async warnonce(*mesg*, ***info*)

class synapse.lib.storm.ScrapeCmd(*runt*, *runtsafe*)

Bases: *Cmd*

Use textual properties of existing nodes to find other easily recognizable nodes.

Examples

Scrape properties from inbound nodes and create standalone nodes. inet:search:query | scrape

Scrape properties from inbound nodes and make refs light edges to the scraped nodes. inet:search:query | scrape –refs

Scrape only the :engine and :text props from the inbound nodes. inet:search:query | scrape :text :engine

Scrape properties inbound nodes and yield newly scraped nodes. inet:search:query | scrape –yield

Skip re-fanging text before scraping. inet:search:query | scrape –skiprefang

Limit scrape to specific forms. inet:search:query | scrape –forms (inet:fqdn, inet:ipv4)

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'scrape'

class synapse.lib.storm.**SleepCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Introduce a delay between returning each result for the storm query.

NOTE: This is mostly used for testing / debugging.

Example

#foo.bar | sleep 0.5

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'sleep'

readonly = True

class synapse.lib.storm.**SpinCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Iterate through all query results, but do not yield any. This can be used to operate on many nodes without returning any.

Example

foo:bar:size=20 [+#hehe] | spin

async execStormCmd(*runt*, *genr*)

Abstract base method

name = 'spin'

readonly = True

class synapse.lib.storm.**StormDmon**

Bases: *Base*

A background storm runtime which is restarted by the cortex.

async bump()

async dmonloop()

pack()

async **run**()

async **stop**()

class synapse.lib.storm.**TagPruneCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Prune a tag (or tags) from nodes.

This command will delete the tags specified as parameters from incoming nodes, as well as all of their parent tags that don't have other tags as children.

For example, given a node with the tags:

#parent #parent.child #parent.child.grandchild

Pruning the parent.child.grandchild tag would remove all tags. If the node had the tags:

#parent #parent.child #parent.child.step #parent.child.grandchild

Pruning the parent.child.grandchild tag will only remove the parent.child.grandchild tag as the parent tags still have other children.

Examples

Prune the parent.child.grandchild tag inet:ipv4=1.2.3.4 | tag.prune parent.child.grandchild

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

hasChildTags(*node*, *tag*)

name = 'tag.prune'

class synapse.lib.storm.**TeeCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute multiple Storm queries on each node in the input stream, joining output streams together.

Commands are executed in order they are given; unless the `--parallel` switch is provided.

Examples

Perform a pivot out and pivot in on a inet:ivp4 node inet:ipv4=1.2.3.4 | tee { -> * } { <- * }

Also emit the inbound node inet:ipv4=1.2.3.4 | tee –join { -> * } { <- * }

Execute multiple enrichment queries in parallel. inet:ipv4=1.2.3.4 | tee -p { enrich.foo } { enrich.bar } { enrich.baz }

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'tee'

async **pipeline**(*runt*, *outq*, *genr=None*)

readonly = True

class synapse.lib.storm.**TreeCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Walk elements of a tree using a recursive pivot.

Examples

pivot upward yielding each FQDN inet:fqdn=www.vertex.link | tree { :domain -> inet:fqdn }

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'tree'

readonly = True

class synapse.lib.storm.**UniqCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Filter nodes by their uniq iden values. When this is used a Storm pipeline, only the first instance of a given node is allowed through the pipeline.

A relative property or variable may also be specified, which will cause this command to only allow through the first node with a given value for that property or value rather than checking the node iden.

Examples

Filter duplicate nodes after pivoting from inet:ipv4 nodes tagged with #badstuff #badstuff +inet:ipv4 ->* | uniq

Unique inet:ipv4 nodes by their :asn property #badstuff +inet:ipv4 | uniq :asn

async **execStormCmd**(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'uniq'

readonly = True

class synapse.lib.storm.**ViewExecCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

Execute a storm query in a different view.

NOTE: Variables are passed through but nodes are not. The behavior of this command may be non-intuitive in relation to the way storm normally operates. For further information on behavior and limitations when using *view.exec*, reference the *view.exec* section of the Synapse User Guide: https://v.vtx.lk/view-exec.

Examples

// Move some tagged nodes to another view inet:fqdn#foo.bar $fqdn=$node.value() | view.exec 95d5f31f0fb414d2b00069d3b1ee64c6 { [inet:fqdn=$fqdn] }

async execStormCmd(*runt*, *genr*)

Abstract base method

getArgParser()

name = 'view.exec'

readonly = True

synapse.lib.storm_format module

class synapse.lib.storm_format.**StormLexer**(*parser*)

Bases: `Lexer`

get_tokens_unprocessed(*text*)

This method should process the text and return an iterable of `(index, tokentype, value)` tuples where `index` is the starting position of the token within the input text.

It must be overridden by subclasses. It is recommended to implement it as a generator to maximize effectiveness.

synapse.lib.storm_format.**highlight_storm**(*parser*, *text*)

Prints a storm query with syntax highlighting

synapse.lib.stormctrl module

exception synapse.lib.stormctrl.**StormBreak**(*item=None*)

Bases: *StormCtrlFlow*

exception synapse.lib.stormctrl.**StormContinue**(*item=None*)

Bases: *StormCtrlFlow*

exception synapse.lib.stormctrl.**StormCtrlFlow**(*item=None*)

Bases: `Exception`

exception synapse.lib.stormctrl.**StormExit**(*item=None*)

Bases: *StormCtrlFlow*

exception synapse.lib.stormctrl.**StormReturn**(*item=None*)

Bases: *StormCtrlFlow*

exception synapse.lib.stormctrl.**StormStop**(*item=None*)

Bases: *StormCtrlFlow*

synapse.lib.stormhttp module

class synapse.lib.stormhttp.**HttpResp**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a HTTP response.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormhttp.**LibHttp**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library exposing an HTTP client API.

For APIs that accept an ssl_opts argument, the dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl_
↪verify argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can be␣
↪included in client_cert.
}
```

async codereason(*code*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns
A key/value pairs.

Return type
dict

async inetHttpConnect(*url*, *headers=None*, *ssl_verify=True*, *timeout=300*, *params=None*, *proxy=None*, *ssl_opts=None*)

strify(*item*)

async urldecode(*text*)

async urlencode(*text*)

class synapse.lib.stormhttp.**WebSocket**

Bases: *Base*, *StormType*

Implements the Storm API for a Websocket.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns
A key/value pairs.

Return type
dict

async **rx**(*timeout=None*)

async **tx**(*mesg*)

synapse.lib.stormsvc module

class synapse.lib.stormsvc.**StormSvc**

Bases: object

The StormSvc mixin class used to make a remote storm service with commands.

async **getStormSvcInfo**()

async **getStormSvcPkgs**()

class synapse.lib.stormsvc.**StormSvcClient**

Bases: *Base*

A StormService is a wrapper for a telepath proxy to a service accessible from the storm runtime.

synapse.lib.stormtypes module

class synapse.lib.stormtypes.**Bool**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a boolean instance.

class synapse.lib.stormtypes.**Bytes**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Bytes object.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**CmdOpts**(*valu*, *path=None*)

Bases: *Dict*

A dictionary like object that holds a reference to a command options namespace. (This allows late-evaluation of command arguments rather than forcing capture)

async **deref**(*name*)

async **iter**()

async **setitem**(*name*, *valu*)

async stormrepr()

async value()

class synapse.lib.stormtypes.**CronJob**(*runt*, *cdef*, *path=None*)

Bases: *Prim*

Implements the Storm api for a cronjob instance.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**Dict**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Dictionary object.

async deref(*name*)

async iter()

async setitem(*name*, *valu*)

async stormrepr()

async value()

class synapse.lib.stormtypes.**Layer**(*runt*, *ldef*, *path=None*)

Bases: *Prim*

Implements the Storm api for a layer instance.

async getEdges()

async getEdgesByN1(*nodeid*)

async getEdgesByN2(*nodeid*)

async getMirrorStatus()

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async getStorNode(*nodeid*)

async getStorNodes()

async liftByProp(*propname*, *propvalu=None*, *propcmpr='='*)

async **liftByTag**(*tagname*, *formname=None*)

async **verify**(*config=None*)

class synapse.lib.stormtypes.**Lib**(*runt*, *name=()*)

Bases: *StormType*

A collection of storm methods under a name

addLibFuncs()

async **deref**(*name*)

async **dyncall**(*iden*, *todo*, *gatekeys=()*)

async **dyniter**(*iden*, *todo*, *gatekeys=()*)

async **initLibAsync**()

async **stormrepr**()

class synapse.lib.stormtypes.**LibAxon**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for interacting with the Cortex's Axon.

For APIs that accept an ssl_opts argument, the dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl␣
↪argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can be␣
↪included in client_cert.
}
```

async **csvrows**(*sha256*, *dialect='excel'*, *errors='ignore'*, ***fmtparams*)

async **del_**(*sha256*)

async **dels**(*sha256s*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **has**(*sha256*)

async **hashset**(*sha256*)

async **jsonlines**(*sha256*, *errors='ignore'*)

async **list**(*offs=0*, *wait=False*, *timeout=None*)

async **metrics**()

async **put**(*byts*)

async **readlines**(*sha256*, *errors='ignore'*)

async **size**(*sha256*)

strify(*item*)

async **upload**(*genr*)

async **urlfile**(**args*, ***kwargs*)

async **wget**(*url*, *headers=None*, *params=None*, *method='GET'*, *json=None*, *body=None*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

async **wput**(*sha256*, *url*, *headers=None*, *params=None*, *method='PUT'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

class synapse.lib.stormtypes.**LibBase**(*runt*, *name=()*)

Bases: *Lib*

The Base Storm Library. This mainly contains utility functionality.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **trycast**(*name*, *valu*)

class synapse.lib.stormtypes.**LibBase64**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for encoding and decoding base64 data.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibBytes**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with bytes storage. This Library is deprecated; use `$lib.axon.*` instead.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibCron**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Cron Jobs in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibCsv**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with csvtool.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibDict**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with dictionaries.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibDmon**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with StormDmons.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibExport**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for exporting data.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async toaxon(query, opts=None)`

`class synapse.lib.stormtypes.LibFeed(runt, name=())`

Bases: *Lib*

A Storm Library for interacting with Cortex feed functions.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.LibGlobals(runt, name)`

Bases: *Lib*

A Storm Library for interacting with global variables which are persistent across the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.LibJsonStor(runt, name=())`

Bases: *Lib*

Implements cortex JSON storage.

`addLibFuncs()`

`async cachedel(path, key)`

`async cacheget(path, key, asof='now', envl=False)`

`async cacheset(path, key, valu)`

`async get(path, prop=None)`

`async has(path)`

`async iter(path=None)`

`async set(path, valu, prop=None)`

class synapse.lib.stormtypes.**LibLayer**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Layers in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibLift**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with lift helpers.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibPipe**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for interacting with non-persistent queues.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibPkg**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Storm Packages.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibPs**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with running tasks on the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibQueue**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with persistent Queues in the Cortex.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibRegx**(*runt*, *name=()*)

Bases: *Lib*

A Storm library for searching/matching with regular expressions.

async escape(*text*)

async findall(*pattern*, *text*, *flags=0*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async matches(*pattern*, *text*, *flags=0*)

async replace(*pattern*, *replace*, *text*, *flags=0*)

async search(*pattern*, *text*, *flags=0*)

class synapse.lib.stormtypes.**LibService**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with Storm Services.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibStr**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with strings.

async concat(**args*)

async format(*text*, ***kwargs*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async join(*sepr*, *items*)

class synapse.lib.stormtypes.**LibTags**(*runt*, *name=()*)

Bases: *Lib*

Storm utility functions for tags.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async prefix(*names*, *prefix*, *ispart=False*)

class synapse.lib.stormtypes.**LibTelepath**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for making Telepath connections to remote services.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**LibTime**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for interacting with timestamps.

async day(*tick*)

async dayofmonth(*tick*)

async dayofweek(*tick*)

async dayofyear(*tick*)

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async hour(tick)`

`async minute(tick)`

`async month(tick)`

`async monthofyear(tick)`

`async second(tick)`

`async toUTC(tick, timezone)`

`async year(tick)`

`class synapse.lib.stormtypes.LibTrigger(runt, name=())`

Bases: `Lib`

A Storm Library for interacting with Triggers in the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.LibVars(runt, name=())`

Bases: `Lib`

A Storm Library for interacting with runtime variables.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.LibView(runt, name=())`

Bases: `Lib`

A Storm Library for interacting with Views in the Cortex.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.List(valu, path=None)`

Bases: `Prim`

Implements the Storm API for a List instance.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`async iter()`

`async setitem(name, valu)`

`async stormrepr()`

`async value()`

`class synapse.lib.stormtypes.Node(node, path=None)`

Bases: `Prim`

Implements the Storm api for a node instance.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.NodeData(node, path=None)`

Bases: `Prim`

A Storm Primitive representing the NodeData stored for a Node.

`async cacheget(name, asof='now')`

`async cacheset(name, valu)`

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.NodeProps(node, path=None)`

Bases: `Prim`

A Storm Primitive representing the properties on a Node.

async get(*name*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns

A key/value pairs.

Return type

dict

async iter()

async list()

async set(*prop*, *valu*)

async setitem(*name*, *valu*)

Set a property on a Node.

Parameters

- **name** (*str*) – The name of the property to set.
- **valu** – The value being set.

Raises

- **s_exc** – NoSuchProp: If the property being set is not valid for the node.
- **s_exc.BadTypeValu** – If the value of the property fails to normalize.

value()

class synapse.lib.stormtypes.**Number**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Number instance.

Storm Numbers are high precision fixed point decimals corresponding to the the hugenum storage type.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns

A key/value pairs.

Return type

dict

async stormrepr()

class synapse.lib.stormtypes.**Path**(*node*, *path=None*)

Bases: *Prim*

Implements the Storm API for the Path object.

getObjLocals()

Get the default list of key-value pairs which may be added to the object .locls dictionary.

Returns

A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.PathMeta(path)`

Bases: *Prim*

Put the storm deref/setitem/iter convention on top of path meta information.

`async deref(name)`

`async iter()`

`async setitem(name, valu)`

`class synapse.lib.stormtypes.PathVars(path)`

Bases: *Prim*

Put the storm deref/setitem/iter convention on top of path variables.

`async deref(name)`

`async iter()`

`async setitem(name, valu)`

`class synapse.lib.stormtypes.Pipe(runt, size)`

Bases: *StormType*

A Storm Pipe provides fast ephemeral queues.

`async close()`

Close the pipe for writing. This will cause the slice()/slices() API to return once drained.

`getObjLocals()`

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

`class synapse.lib.stormtypes.Prim(valu, path=None)`

Bases: *StormType*

The base type for all Storm primitive values.

`async bool()`

`async iter()`

`async nodes()`

`async stormrepr()`

`value()`

class synapse.lib.stormtypes.**Proxy**(*runt*, *proxy*, *path=None*)

Bases: *StormType*

Implements the Storm API for a Telepath proxy.

These can be created via `$lib.telepath.open()`. Storm Service objects are also Telepath proxy objects.

Methods called off of these objects are executed like regular Telepath RMI calls.

An example of calling a method which returns data:

```
$prox = $lib.telepath.open($url)
$result = $prox.doWork($data)
return ( $result )
```

An example of calling a method which is a generator:

```
$prox = $lib.telepath.open($url)
for $item in $prox.genrStuff($data) {
    $doStuff($item)
}
```

async deref(*name*)

async stormrepr()

class synapse.lib.stormtypes.**ProxyGenrMethod**(*meth*, *path=None*)

Bases: *StormType*

Implements the generator methods for the telepath:proxy.

An example of calling a method which is a generator:

```
$prox = $lib.telepath.open($url)
for $item in $prox.genrStuff($data) {
    $doStuff($item)
}
```

async stormrepr()

class synapse.lib.stormtypes.**ProxyMethod**(*runt*, *meth*, *path=None*)

Bases: *StormType*

Implements the call methods for the telepath:proxy.

An example of calling a method which returns data:

```
$prox = $lib.telepath.open($url)
$result = $prox.doWork($data)
$doStuff($result)
```

async stormrepr()

class synapse.lib.stormtypes.**Query**(*text*, *varz*, *runt*, *path=None*)

Bases: *Prim*

A storm primitive representing an embedded query.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async iter()

async nodes()

async stormrepr()

class synapse.lib.stormtypes.**Queue**(*runt*, *name*, *info*)

Bases: *StormType*

A StormLib API instance of a named channel in the Cortex multiqueue.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async stormrepr()

class synapse.lib.stormtypes.**Service**(*runt*, *ssvc*)

Bases: *Proxy*

async deref(*name*)

class synapse.lib.stormtypes.**Set**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a Set object.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async iter()

async stormrepr()

class synapse.lib.stormtypes.**StormHiveDict**(*runt*, *info*)

Bases: *Prim*

A Storm Primitive representing a HiveDict.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async iter()

value()

class synapse.lib.stormtypes.StormType(*path=None*)

Bases: `object`

The base type for storm runtime value objects.

async deref(*name*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

ismutable()

async setitem(*name*, *valu*)

class synapse.lib.stormtypes.StormTypesRegistry

Bases: `object`

addStormLib(*path*, *ctor*)

addStormType(*path*, *ctor*)

base_undefined_types = ('any', 'int', 'lib', 'null', 'time', 'prim', 'undef', 'float', 'generator')

delStormLib(*path*)

delStormType(*path*)

getLibDocs(*lib=None*)

getTypeDocs(*styp: str = None*)

iterLibs()

iterTypes()

known_types = {'aha:pool', 'auth:gate', 'auth:role', 'auth:user', 'auth:user:json', 'auth:user:profile', 'auth:user:vars', 'boolean', 'bytes', 'cache:fixed', 'cmdopts', 'cronjob', 'dict', 'hive:dict', 'http:api', 'http:api:methods', 'http:api:perms', 'http:api:request', 'http:api:request:headers', 'http:api:vars', 'inet:http:oauth:v1:client', 'inet:http:resp', 'inet:http:socket', 'inet:imap:server', 'inet:smtp:message', 'json:schema', 'layer', 'list', 'model:form', 'model:property', 'model:tagprop', 'model:type', 'node', 'node:data', 'node:path', 'node:path:meta', 'node:path:vars', 'node:props', 'number', 'pipe', 'proj:comment', 'proj:comments', 'proj:epic', 'proj:epics', 'proj:project', 'proj:sprint', 'proj:sprints', 'proj:ticket', 'proj:tickets', 'queue', 'set', 'spooled:set', 'stat:tally', 'stix:bundle', 'storm:query', 'str', 'telepath:proxy', 'telepath:proxy:genrmethod', 'telepath:proxy:method', 'text', 'trigger', 'vault', 'vault:data', 'view', 'xml:element'}

registerLib(*ctor*)

Decorator to register a StormLib

registerType(*ctor*)

Decorator to register a StormPrim

rtypes = {}

undefined_types = {'any', 'float', 'generator', 'int', 'lib', 'null', 'prim', 'time', 'undef'}

class synapse.lib.stormtypes.**Str**(*valu*, *path=None*)

Bases: *Prim*

Implements the Storm API for a String object.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**Text**(*valu*, *path=None*)

Bases: *Prim*

A mutable text type for simple text construction.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

class synapse.lib.stormtypes.**Trigger**(*runt*, *tdef*)

Bases: *Prim*

Implements the Storm API for a Trigger.

async **deref**(*name*)

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **move**(*viewiden*)

async **pack**()

async **set**(*name*, *valu*)

class synapse.lib.stormtypes.**Undef**

Bases: `object`

async **stormrepr**()

class synapse.lib.stormtypes.**View**(*runt*, *vdef*, *path=None*)

Bases: *Prim*

Implements the Storm api for a View instance.

async **addNode**(*form*, *valu*, *props=None*)

async **delMergeRequest**()

async **delMergeVote**(*useriden=None*)

async **detach**()

async **getMergeRequest**()

async **getMergeRequestSummary**()

async **getMerges**()

async **getMergingViews**()

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

async **setMergeComment**(*comment*)

async **setMergeRequest**(*comment=None*)

async **setMergeVote**(*approved=True*, *comment=None*)

async **setMergeVoteComment**(*comment*)

async **viewDynCall**(*todo*, *perm*)

async **viewDynIter**(*todo*, *perm*)

synapse.lib.stormtypes.**allowed**(*perm*, *gateiden=None*)

synapse.lib.stormtypes.**allowedEasyPerm**(*item*, *perm*)

synapse.lib.stormtypes.**confirm**(*perm*, *gateiden=None*)

synapse.lib.stormtypes.**confirmEasyPerm**(*item*, *perm*, *mesg=None*)

synapse.lib.stormtypes.**fromprim**(*valu*, *path=None*, *basetypes=True*)

synapse.lib.stormtypes.**getCallSig**(*func*) → Signature

Get the callsig of a function, stripping self if present.

synapse.lib.stormtypes.**getDoc**(*obj*, *errstr*)

Helper to get __doc__

synapse.lib.stormtypes.**intify**(*x*)

synapse.lib.stormtypes.**ismutable**(*valu*)

async synapse.lib.stormtypes.**kwarg_format**(*_text*, ***kwargs*)

Replaces instances curly-braced argument names in text with their values

synapse.lib.stormtypes.**stormfunc**(*readonly=False*)

async synapse.lib.stormtypes.**tobool**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**tobuidhex**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**tocmprvalu**(*valu*)

async synapse.lib.stormtypes.**toint**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**toiter**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**tonumber**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**toprim**(*valu*, *path=None*)

async synapse.lib.stormtypes.**torepr**(*valu*, *usestr=False*)

async synapse.lib.stormtypes.**tostor**(*valu*)

async synapse.lib.stormtypes.**tostr**(*valu*, *noneok=False*)

async synapse.lib.stormtypes.**totype**(*valu*, *basetypes=False*) → str

Convert a value to its Storm type string.

Parameters

- **valu** – The object to check.
- **basetypes** (*bool*) – If True, return the base Python class name as a fallback.

Returns

The type name.

Return type

str

Raises

StormRuntimeError – If the valu does not resolve to a known type and basetypes=False.

async synapse.lib.stormtypes.**typeerr**(*name*, *reqt*)

synapse.lib.stormwhois module

class synapse.lib.stormwhois.**LibWhois**(*runt*, *name=()*)

Bases: *Lib*

A Storm Library for providing a consistent way to generate guids for WHOIS / Registration Data in Storm.

getObjLocals()

Get the default list of key-value pairs which may be added to the object `.locls` dictionary.

Returns
A key/value pairs.

Return type
dict

synapse.lib.structlog module

class synapse.lib.structlog.**JsonFormatter**(**args*, ***kwargs*)

Bases: `Formatter`

format(*record: LogRecord*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

synapse.lib.task module

class synapse.lib.task.**Task**

Bases: *Base*

The synapse Task object implements concepts similar to process trees for asyncio.Task instances.

async **kill**()

pack()

async **worker**(*coro*, *name='worker'*)

synapse.lib.task.**current**()

Return the current synapse task.

async synapse.lib.task.**executor**(*func*, **args*, ***kwargs*)

Execute a function in an executor thread.

Parameters
todo (*(func,args,kwargs)*) – A todo tuple.

synapse.lib.task.**loop**()

synapse.lib.task.**user**()

Return the current task user.

`synapse.lib.task.`**`username`**`()`

Return the current task user name.

`synapse.lib.task.`**`vardefault`**`(`*name*, *func*`)`

Add a default constructor for a particular task-local variable

All future calls to taskVarGet with the same name will return the result of calling func

`synapse.lib.task.`**`varget`**`(`*name*, *defval=None*, *task=None*`)`

Access a task local variable by name

Precondition:
If task is None, this must be called from task context

`synapse.lib.task.`**`varinit`**`(`*task=None*`)`

Initializes (or re-initializes for testing purposes) all of a task's task-local variables

Precondition:
If task is None, this must be called from task context

`synapse.lib.task.`**`varset`**`(`*name*, *valu*, *task=None*`)`

Set a task-local variable

Parameters
task – If task is None, uses current task

Precondition:
If task is None, this must be called from task context

synapse.lib.thishost module

`synapse.lib.thishost.`**`get`**`(`*prop*`)`

Retrieve a property from the hostinfo dictionary.

Example

import synapse.lib.thishost as s_thishost

if s_thishost.get('platform') == 'windows':
dostuff()

`synapse.lib.thishost.`**`hostaddr`**`(`*dest='8.8.8.8'*`)`

Retrieve the ipv4 address for this host (optionally as seen from dest). .. rubric:: Example

addr = s_socket.hostaddr()

synapse.lib.thisplat module

synapse.lib.threads module

`synapse.lib.threads.`**`current`**`()`

`synapse.lib.threads.`**`iden`**`()`

synapse.lib.time module

Time related utilities for synapse "epoch millis" time values.

synapse.lib.time.**day**(*tick*)

synapse.lib.time.**dayofmonth**(*tick*)

synapse.lib.time.**dayofweek**(*tick*)

synapse.lib.time.**dayofyear**(*tick*)

synapse.lib.time.**delta**(*text*)

Parse a simple time delta string and return the delta.

synapse.lib.time.**hour**(*tick*)

synapse.lib.time.**ival**(**times*)

synapse.lib.time.**minute**(*tick*)

synapse.lib.time.**month**(*tick*)

synapse.lib.time.**parse**(*text*, *base=None*, *chop=False*)

Parse a time string into an epoch millis value.

Parameters

- **text** (*str*) – Time string to parse
- **base** (*int or None*) – Milliseconds to offset the time from
- **chop** (*bool*) – Whether to chop the digit-only string to 17 chars

Returns

Epoch milliseconds

Return type

int

synapse.lib.time.**parsetz**(*text*)

Parse timezone from time string, with UTC as the default.

Parameters

text (*str*) – Time string

Returns

A tuple of text with tz chars removed and base milliseconds to offset time.

Return type

tuple

synapse.lib.time.**repr**(*tick*, *pack=False*)

Return a date string for an epoch-millis timestamp.

Parameters

tick (*int*) – The timestamp in milliseconds since the epoch.

Returns

A date time string

Return type

(str)

synapse.lib.time.**second**(*tick*)

synapse.lib.time.**toUTC**(*tick*, *fromzone*)

synapse.lib.time.**wildrange**(*text*)

Parse an interval from a wild card time stamp: 2021/10/31*

synapse.lib.time.**year**(*tick*)

synapse.lib.trigger module

class synapse.lib.trigger.**Trigger**(*view*, *tdef*)

Bases: object

async **execute**(*node*, *vars=None*)

Actually execute the query

get(*name*)

getStorNode(*form*)

pack()

async **set**(*name*, *valu*)

Set one of the dynamic elements of the trigger definition.

class synapse.lib.trigger.**Triggers**(*view*)

Bases: object

Manages "triggers", conditions where changes in data result in new storm queries being executed.

Note: These methods should not be called directly under normal circumstances. Use the owning "View" object to ensure that mirrors/clusters members get the same changes.

get(*iden*)

list()

async **load**(*tdef*)

pop(*iden*)

async **runEdgeAdd**(*n1*, *verb*, *n2*)

async **runEdgeDel**(*n1*, *verb*, *n2*)

async **runNodeAdd**(*node*)

async **runNodeDel**(*node*)

async **runPropSet**(*node*, *prop*, *oldv*)

async **runTagAdd**(*node*, *tag*)

async **runTagDel**(*node*, *tag*)

synapse.lib.trigger.**reqValidTdef**(*conf*)

synapse.lib.types module

class synapse.lib.types.**Array**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

isarray = True

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class synapse.lib.types.**Bool**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 2

class synapse.lib.types.**Comp**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

getCompOffs(*name*)

If this type is a compound, return the field offset for the given property name or None.

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 13

class synapse.lib.types.**Data**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

norm(*valu*)

Normalize the value for a given type.

Parameters
valu (*obj*) – The value to normalize.

Returns
The normalized valu, info tuple.

Return type
((obj,dict))

Notes

The info dictionary uses the following key conventions:
subs (dict): The normalized sub-fields as name: valu entries.

postTypeInit()

stortype: int = 13

class synapse.lib.types.**Duration**(*modl*, *name*, *info*, *opts*)

Bases: *IntBase*

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 5

class synapse.lib.types.**Edge**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

getCompOffs(*name*)

If this type is a compound, return the field offset for the given property name or None.

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 13

class synapse.lib.types.**FieldHelper**(*modl*, *tname*, *fields*)

Bases: `defaultdict`

Helper for Comp types. Performs Type lookup/creation upon first use.

class synapse.lib.types.**Float**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 22

class synapse.lib.types.**Guid**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

stortype: int = 10

class synapse.lib.types.**Hex**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

stortype: int = 1

class synapse.lib.types.**HugeNum**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

norm(*valu*)

Normalize the value for a given type.

Parameters
valu (*obj*) – The value to normalize.

Returns
The normalized valu, info tuple.

Return type
((obj,dict))

Notes

The info dictionary uses the following key conventions:
subs (dict): The normalized sub-fields as name: valu entries.

stortype: int = 23

class synapse.lib.types.**Int**(*modl*, *name*, *info*, *opts*)

Bases: *IntBase*

merge(*oldv*, *newv*)

Allow types to "merge" data from two sources based on value precedence.

Parameters

- **valu** (*object*) – The current value.
- **newv** (*object*) – The updated value.

Returns
The merged value.

Return type
(object)

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class synapse.lib.types.**IntBase**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

class synapse.lib.types.**Ival**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

An interval, i.e. a range, of times

merge(*oldv*, *newv*)

Allow types to "merge" data from two sources based on value precedence.

Parameters

- **valu** (*object*) – The current value.
- **newv** (*object*) – The updated value.

Returns

The merged value.

Return type

(object)

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 12

class synapse.lib.types.**Loc**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stems(*valu*)

stortype: int = 15

class synapse.lib.types.**Ndef**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 13

class synapse.lib.types.**NodeProp**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

stortype: int = 13

class synapse.lib.types.**Range**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 13

class synapse.lib.types.**Str**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 1

class synapse.lib.types.**Tag**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.lib.types.**TagPart**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.lib.types.**Taxon**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.lib.types.**Taxonomy**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class synapse.lib.types.**Time**(*modl*, *name*, *info*, *opts*)

Bases: *IntBase*

getTickTock(*vals*)

Get a tick, tock time pair.

Parameters
vals (*list*) – A pair of values to norm.

Returns
A ordered pair of integers.

Return type
(int, int)

merge(*oldv*, *newv*)

Allow types to "merge" data from two sources based on value precedence.

Parameters

- **valu** (*object*) – The current value.
- **newv** (*object*) – The updated value.

Returns
The merged value.

Return type
(object)

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 11

class synapse.lib.types.**TimeEdge**(*modl*, *name*, *info*, *opts*)

Bases: *Edge*

getCompOffs(*name*)

If this type is a compound, return the field offset for the given property name or None.

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 13

class synapse.lib.types.**Type**(*modl*, *name*, *info*, *opts*)

Bases: `object`

clone(*opts*)

Create a new instance of this type with the specified options.

Parameters
opts (*dict*) – The type specific options for the new instance.

cmpr(*val1*, *name*, *val2*)

Compare the two values using the given type specific comparator.

extend(*name*, *opts*, *info*)

Extend this type to construct a sub-type.

Parameters

- **name** (*str*) – The name of the new sub-type.
- **opts** (*dict*) – The type options for the sub-type.
- **info** (*dict*) – The type info for the sub-type.

Returns
A new sub-type instance.

Return type
(synapse.types.Type)

getCmprCtor(*name*)

getCompOffs(*name*)

If this type is a compound, return the field offset for the given property name or None.

getLiftHintCmpr(*valu*, *cmpr*)

getLiftHintCmprCtor(*name*)

getStorCmprs(*cmpr*, *valu*)

getStorNode(*form*)

getTypeDef()

getTypeVals(*valu*)

isarray = False

merge(*oldv*, *newv*)

Allow types to "merge" data from two sources based on value precedence.

Parameters

- **valu** (*object*) – The current value.
- **newv** (*object*) – The updated value.

Returns

The merged value.

Return type

(object)

norm(*valu*)

Normalize the value for a given type.

Parameters

valu (*obj*) – The value to normalize.

Returns

The normalized valu, info tuple.

Return type

((obj,dict))

Notes

The info dictionary uses the following key conventions:

subs (dict): The normalized sub-fields as name: valu entries.

pack()

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

setCmprCtor(*name*, *func*)

Set a comparator ctor for a given named comparison operation.

Parameters

- **name** (*str*) – Name of the comparison operation.
- **func** – Function which returns a comparator.

Notes

Comparator ctors should expect to get the right-hand-side of the comparison as their argument, and the returned function should expect to get the left hand side of the comparison and return a boolean from there.

setLiftHintCmprCtor(*name*, *func*)

setNormFunc(*typo*, *func*)

Register a normalizer function for a given python type.

Parameters

- **typo** (*type*) – A python type/class to normalize.
- **func** (*function*) – A callback which normalizes a python value.

stortype: int = None

class synapse.lib.types.**Velocity**(*modl*, *name*, *info*, *opts*)

Bases: *IntBase*

oflight = 299792458000

postTypeInit()

stortype: int = 9

synapse.lib.urlhelp module

synapse.lib.urlhelp.**chopurl**(*url*)

A sane "stand alone" url parser.

Example

info = chopurl(url)

synapse.lib.urlhelp.**sanitizeUrl**(*url*)

Returns a URL with the password (if present) replaced with ****

RFC 3986 3.2.1 'Applications should not render as clear text any data after the first colon (":") character found within a userinfo subcomponent unless the data after the colon is the empty string (indicating no password)'

Essentially, replace everything between the 2nd colon (if it exists) and the first succeeding at sign. Return the original string otherwise.

Note: this depends on this being a reasonably-well formatted URI that starts with a scheme (e.g. http) and '//:' Failure of this condition yields the original string.

synapse.lib.version module

Synapse utilites for dealing with Semvar versioning. This includes the Synapse version information.

synapse.lib.version.**fmtVersion**(**vsnparts*)

Join a string of parts together with a . separator.

Parameters
***vsnparts** –

Returns:

synapse.lib.version.**matches**(*vers*, *cmprvers*)

Check if a version string matches a version comparison string.

synapse.lib.version.**packVersion**(*major*, *minor=0*, *patch=0*)

Pack a set of major/minor/patch integers into a single integer for storage.

Parameters

- **major** (*int*) – Major version level integer.
- **minor** (*int*) – Minor version level integer.
- **patch** (*int*) – Patch version level integer.

Returns
System normalized integer value to represent a software version.

Return type
int

synapse.lib.version.**parseSemver**(*text*)

Parse a Semantic Version string into is component parts.

Parameters

- **text** (*str*) – A text string to parse into semver components. This string has whitespace and leading 'v'
- **it.** (*characters stripped off of*) –

Examples

Parse a string into it semvar parts:

```
parts = parseSemver('v1.2.3')
```

Returns
The dictionary will contain the keys 'major', 'minor' and 'patch' pointing to integer values. The dictionary may also contain keys for 'build' and 'pre' information if that data is parsed out of a semver string. None is returned if the string is not a valid Semver string.

Return type
dict

synapse.lib.version.**parseVersionParts**(*text*, *seps=('.', '-', '_', '+')*)

Extract a list of major/minor/version integer strings from a string.

Parameters

- **text** (*str*) – String to parse
- **seps** (*tuple*) – A tuple or list of separators to use when parsing the version string.

Examples

Parse a simple version string into a major and minor parts:

```
parts = parseVersionParts('1.2')
```

Parse a complex version string into a major and minor parts:

```
parts = parseVersionParts('wowsoft_1.2')
```

Parse a simple version string into a major, minor and patch parts. Parts after the "3." are dropped from the results:

```
parts = parseVersionParts('1.2.3.4.5')
```

Notes

This attempts to brute force out integers from the version string by stripping any leading ascii letters and part separators, and then regexing out numeric parts optionally followed by part separators. It will stop at the first mixed-character part encountered. For example, "1.2-3a" would only parse out the "1" and "2" from the string.

Returns
Either a empty dictionary or dictionary containing up to three keys, 'major', 'minor' and 'patch'.

Return type
dict

synapse.lib.version.**reqVersion**(*valu*, *reqver*, *exc=<class 'synapse.exc.BadVersion'>*, *mesg='Provided version does not match required version.'*)

Require a given version tuple is valid for a given requirements string.

Parameters

- **Optional[Tuple[int** (*valu*) – Major, minor and patch value to check.
- **int** – Major, minor and patch value to check.
- **int]]** – Major, minor and patch value to check.
- **reqver** (*str*) – A requirements version string.
- **exc** (*s_exc.SynErr*) – The synerr class to raise.
- **mesg** (*str*) – The message to pass in the exception.

Returns
If the value is in bounds of minver and maxver.

Return type
None

Raises
s_exc.BadVersion – If a precondition is incorrect or a version value is out of bounds.

`synapse.lib.version.unpackVersion(ver)`

Unpack a system normalized integer representing a softare version into its component parts.

Parameters
ver (*int*) – System normalized integer value to unpack into a tuple.

Returns
A tuple containing the major, minor and patch values shifted out of the integer.

Return type
(int, int, int)

synapse.lib.view module

`class synapse.lib.view.View`

Bases: *Pusher*

A view represents a cortex as seen from a specific set of layers.

The view class is used to implement Copy-On-Write layers as well as interact with a subset of the layers configured in a Cortex.

`async addLayer(layriden, indx=None)`

`async addNode(form, valu, props=None, user=None)`

`async addNodeEdits(edits, meta)`

A telepath compatible way to apply node edits to a view.

NOTE: This does cause trigger execution.

`async addTrigQueue(triginfo, nexsitem)`

`async addTrigger(tdef)`

Adds a trigger to the view.

`async callStorm(text, opts=None)`

`async callStormIface(name, todo)`

`async delMergeRequest()`

`async delMergeVote(useriden)`

`async delTrigQueue(offs)`

`async delTrigger(iden)`

`async delete()`

Delete the metadata for this view.

Note: this does not delete any layer storage.

`async detach()`

Detach the view from its parent but do not change the layers. (this is not reversible!)

`async eval(text, opts=None, log_info=None)`

Evaluate a storm query and yield Nodes only.

async **finiMergeTask**()

async **finiTrigTask**()

async **fork**(*ldef=None*, *vdef=None*)

Make a new view inheriting from this view with the same layers and a new write layer on top

Parameters

- **ldef** – layer parameter dict
- **vdef** – view parameter dict
- **cortex.addLayer** (`Passed through to`) –

Returns

new view object, with an iden the same as the new write layer iden

async **getEdgeVerbs**()

async **getEdges**(*verb=None*)

async **getFormCounts**()

getMergeRequest()

async **getMergeVotes**()

async **getMerges**()

Yield the historical merges into this view.

async **getMergingViews**()

async **getPropArrayCount**(*propname*, *valu=<synapse.common.NoValu object>*)

async **getPropCount**(*propname*, *valu=<synapse.common.NoValu object>*)

async **getStorNodes**(*buid*)

Return a list of storage nodes for the given buid in layer order.

async **getTagPropCount**(*form*, *tag*, *propname*, *valu=<synapse.common.NoValu object>*)

async **getTrigger**(*iden*)

hasKids()

init2()

We have a second round of initialization so the views can get a handle to their parents which might not be initialized yet

async **initMergeTask**()

async **initTrigTask**()

isForkOf(*viewiden*)

async **isMergeReady**()

isafork()

async **iterStormPodes**(*text*, *opts=None*) async **listTriggers**()

List all the triggers in the view.

async **merge**(*useriden=None*, *force=False*)

Merge this view into its parent. All changes made to this view will be applied to the parent. Parent's triggers will be run.

async **mergeAllowed**(*user=None*, *force=False*)

Check whether a user can merge a view into its parent.

NOTE: This API may not be used to check for merges based on quorum votes.

async **mergeStormIface**(*name*, *todo*)

Allow an interface which specifies a generator use case to yield (priority, value) tuples and merge results from multiple generators yielded in ascending priority order.

async **nodes**(*text*, *opts=None*)

A simple non-streaming way to return a list of nodes.

async **pack**()

reqNoParentQuorum()

reqParentQuorum()

reqValidVoter(*useriden*)

async **runEdgeAdd**(*n1*, *edge*, *n2*)

async **runEdgeDel**(*n1*, *edge*, *n2*)

async **runNodeAdd**(*node*)

async **runNodeDel**(*node*)

async **runPropSet**(*node*, *prop*, *oldv*)

Handle when a prop set trigger event fired

async **runTagAdd**(*node*, *tag*, *valu*)

async **runTagDel**(*node*, *tag*, *valu*)

async **runViewMerge**()

async **scrapeIface**(*text*, *unique=False*, *refang=True*)

async **setLayers**(*layers*)

Set the view layers from a list of idens. NOTE: view layers are stored "top down" (the write layer is self.layers[0])

async **setMergeComment**(*comment*)

async **setMergeRequest**(*mergeinfo*)

async **setMergeVote**(*vote*)

async **setMergeVoteComment**(*useriden*, *comment*)

async **setTriggerInfo**(*iden*, *name*, *valu*)

async setViewInfo(*name*, *valu*)
Set a mutable view property.

async snap(*user*)

async classmethod snapctor(**args*, ***kwargs*)

async storNodeEdits(*edits*, *meta*)

async storm(*text*, *opts=None*)
Evaluate a storm query and yield result messages. :Yields: *((str,dict))* – Storm messages.

async stormlist(*text*, *opts=None*)

async tryToMerge(*tick*)

async wipeAllowed(*user=None*)
Check whether a user can wipe the write layer in the current view.

async wipeLayer(*useriden=None*)
Delete the data in the write layer by generating del nodeedits. Triggers will be run.

class synapse.lib.view.**ViewApi**
Bases: *CellApi*

async getCellIden()

async getEditSize()

saveNodeEdits(*edits*, *meta*)

async storNodeEdits(*edits*, *meta*)

async syncNodeEdits2(*offs*, *wait=True*)

synapse.lookup package

Submodules

synapse.lookup.cvss module

synapse.lookup.iana module

synapse.lookup.iso3166 module

Provides data for the ISO 3166-1 Country codes.

Reference:
https://en.wikipedia.org/wiki/ISO_3166

synapse.lookup.iso3166.**makeColLook**(*rows*, *scol*, *dcol*)

synapse.lookup.macho module

synapse.lookup.macho.**getLoadCmdTypes**()

synapse.lookup.macho.**getSectionTypes**()

synapse.lookup.pe module

synapse.lookup.pe.**getLangCodes**()

synapse.lookup.pe.**getRsrcTypes**()

synapse.lookup.phonenum module

synapse.lookup.phonenum.**formPhoneNode**(*node*, *valu*)

synapse.lookup.phonenum.**getPhoneInfo**(*numb*)

Walk the phone info tree to find the best-match info for the given number.

Example

info = getPhoneInfo(17035551212) country = info.get('cc')

synapse.lookup.phonenum.**initPhoneTree**()

synapse.lookup.phonenum.**phnode**(*valu*)

synapse.lookup.timezones module

Timezones are defined per RFC822 5.1 (plus GMT and UTC), with values representing offsets from UTC in milliseconds.

synapse.lookup.timezones.**getTzNames**()

Return a tuple of all supported timezone names.

synapse.lookup.timezones.**getTzOffset**(*name*, *defval=None*)

Return tuple of the UTC offset in milliseconds and an info dict.

synapse.models package

Subpackages

synapse.models.gov package

Submodules

synapse.models.gov.cn module

class synapse.models.gov.cn.**GovCnModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.gov.intl module

class synapse.models.gov.intl.**GovIntlModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.gov.us module

class synapse.models.gov.us.**GovUsModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

Submodules

synapse.models.auth module

class synapse.models.auth.**AuthModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.base module

class synapse.models.base.**BaseModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.belief module

class synapse.models.belief.**BeliefModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.biz module

class synapse.models.biz.**BizModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

synapse.models.crypto module

class synapse.models.crypto.**CryptoModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

synapse.models.dns module

class synapse.models.dns.**DnsModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

class synapse.models.dns.**DnsName**(*modl*, *name*, *info*, *opts*)

Bases: `Str`

postTypeInit()

synapse.models.economic module

class synapse.models.economic.**EconModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

synapse.models.files module

class synapse.models.files.**FileBase**(*modl*, *name*, *info*, *opts*)

Bases: `Str`

postTypeInit()

class synapse.models.files.**FileBytes**(*modl*, *name*, *info*, *opts*)

Bases: `Str`

postTypeInit()

class synapse.models.files.**FileModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

async initCoreModule()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

class `synapse.models.files.`**`FilePath`**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

`postTypeInit()`

synapse.models.geopol module

class `synapse.models.geopol.`**`PolModule`**(*core*, *conf=None*)

Bases: *CoreModule*

`getModelDefs()`

synapse.models.geospace module

class `synapse.models.geospace.`**`Area`**(*modl*, *name*, *info*, *opts*)

Bases: *Int*

`postTypeInit()`

`repr`(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class `synapse.models.geospace.`**`Dist`**(*modl*, *name*, *info*, *opts*)

Bases: *Int*

`postTypeInit()`

`repr`(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class `synapse.models.geospace.`**`GeoModule`**(*core*, *conf=None*)

Bases: *CoreModule*

`getModelDefs()`

class `synapse.models.geospace.`**`LatLong`**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

`postTypeInit()`

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 14

synapse.models.inet module

class synapse.models.inet.**Addr**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.models.inet.**Cidr4**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.models.inet.**Cidr6**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.models.inet.**Email**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

class synapse.models.inet.**Fqdn**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 17

class synapse.models.inet.**HttpCookie**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

getTypeVals(*valu*)

class synapse.models.inet.**IPv4**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

The base type for an IPv4 address.

getCidrRange(*text*)

getNetRange(*text*)

getTypeVals(*valu*)

postTypeInit()

repr(*norm*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 4

class synapse.models.inet.**IPv4Range**(*modl*, *name*, *info*, *opts*)

Bases: *Range*

postTypeInit()

class synapse.models.inet.**IPv6**(*modl*, *name*, *info*, *opts*)

Bases: *Type*

getCidrRange(*text*)

getNetRange(*text*)

getTypeVals(*valu*)

postTypeInit()

stortype: int = 18

class synapse.models.inet.**IPv6Range**(*modl*, *name*, *info*, *opts*)

Bases: *Range*

postTypeInit()

class synapse.models.inet.**InetModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

async initCoreModule()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

class synapse.models.inet.**Rfc2822Addr**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

An RFC 2822 compatible email address parser

postTypeInit()

class synapse.models.inet.**Url**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

synapse.models.inet.**getAddrScope**(*ipv6*)

synapse.models.inet.**getAddrType**(*ip*)

synapse.models.infotech module

class synapse.models.infotech.**Cpe22Str**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

CPE 2.2 Formatted String https://cpe.mitre.org/files/cpe-specification_2.2.pdf

class synapse.models.infotech.**Cpe23Str**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

CPE 2.3 Formatted String

```
https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7695.pdf

(Section 6.2)

cpe:2.3: part : vendor : product : version : update : edition :
    language : sw_edition : target_sw : target_hw : other

* = "any"
- = N/A
```

class synapse.models.infotech.**ItModule**(*core*, *conf=None*)

Bases: *CoreModule*

bruteVersionStr(*valu*)

This API is deprecated.

Brute force the version out of a string.

Parameters
valu (*str*) – String to attempt to get version information for.

Notes

This first attempts to parse strings using the it:semver normalization before attempting to extract version parts out of the string.

Returns
The system normalized version integer and a subs dictionary.

Return type
int, dict

getModelDefs()

async initCoreModule()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

class synapse.models.infotech.SemVer(*modl*, *name*, *info*, *opts*)

Bases: *Int*

Provides support for parsing a semantic version string into its component parts. This normalizes a version string into an integer to allow version ordering. Prerelease information is disregarded for integer comparison purposes, as we cannot map an arbitrary pre-release version into a integer value

Major, minor and patch levels are represented as integers, with a max width of 20 bits. The comparable integer value representing the semver is the bitwise concatenation of the major, minor and patch levels.

Prerelease and build information will be parsed out and available as strings if that information is present.

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

synapse.models.infotech.**chopCpe22**(*text*)

CPE 2.2 Formatted String https://cpe.mitre.org/files/cpe-specification_2.2.pdf

synapse.models.infotech.**cpesplit**(*text*)

synapse.models.infotech.**zipCpe22**(*parts*)

synapse.models.language module

class synapse.models.language.LangModule(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.material module

A data model focused on material objects.

class synapse.models.material.**MatModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.media module

class synapse.models.media.**MediaModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.orgs module

class synapse.models.orgs.**OuModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.person module

class synapse.models.person.**PsModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.proj module

class synapse.models.proj.**ProjectModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

async initCoreModule()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

synapse.models.risk module

class synapse.models.risk.**CvssV2**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

class synapse.models.risk.**CvssV3**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

class synapse.models.risk.**RiskModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.science module

class synapse.models.science.**ScienceModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.syn module

class synapse.models.syn.**SynModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

initCoreModule()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

synapse.models.telco module

class synapse.models.telco.**Imei**(*modl*, *name*, *info*, *opts*)

Bases: *Int*

postTypeInit()

class synapse.models.telco.**Imsi**(*modl*, *name*, *info*, *opts*)

Bases: *Int*

postTypeInit()

class synapse.models.telco.**Phone**(*modl*, *name*, *info*, *opts*)

Bases: *Str*

postTypeInit()

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

class synapse.models.telco.**TelcoModule**(*core*, *conf=None*)

Bases: *CoreModule*

getModelDefs()

synapse.models.telco.**chop_imei**(*imei*)

synapse.models.telco.**digits**(*text*)

synapse.models.telco.**imeicsum**(*text*)

Calculate the imei check byte.

synapse.models.transport module

class synapse.models.transport.**TransportModule**(*core*, *conf=None*)

Bases: CoreModule

getModelDefs()

synapse.servers package

Submodules

synapse.servers.aha module

synapse.servers.axon module

synapse.servers.cell module

async synapse.servers.cell.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.servers.cortex module

synapse.servers.cryotank module

synapse.servers.jsonstor module

synapse.servers.stemcell module

synapse.servers.stemcell.**getStemCell**(*dirn*)

async synapse.servers.stemcell.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tests package

Submodules

synapse.tests.nopmod module

A python module used for testing dmon dynamic module loading.

synapse.tests.utils module

This contains the core test helper code used in Synapse.

This gives the opportunity for third-party users of Synapse to test their code using some of the same helpers used to test Synapse.

The core class, synapse.tests.utils.SynTest is a subclass of unittest.TestCase, with several wrapper functions to allow for easier calls to assert* functions, with less typing. There are also Synapse specific helpers, to load Cortexes and whole both multi-component environments into memory.

Since SynTest is built from unittest.TestCase, the use of SynTest is compatible with the unittest, nose and pytest frameworks. This does not lock users into a particular test framework; while at the same time allowing base use to be invoked via the built-in Unittest library, with one important exception: due to an unfortunate design approach, you cannot use the unittest module command line to run a *single* async unit test. pytest works fine though.

class synapse.tests.utils.**AsyncStreamEvent**(**args*, ***kwargs*)

Bases: `StringIO`, `Event`

A combination of a io.StringIO object and an asyncio.Event object.

setMesg(*mesg*)

Clear the internal event and set a new message that is used to set the event.

Parameters
mesg (*str*) – The string to monitor for.

Returns
None

async **wait**(*timeout=None*)

Block until the internal flag is true.

If the internal flag is true on entry, return True immediately. Otherwise, block until another coroutine calls set() to set the flag to true, then return True.

write(*s*)

Write string to file.

Returns the number of characters written, which is always equal to the length of the string.

class synapse.tests.utils.**CmdGenerator**(*cmds*)

Bases: `object`

addCmd(*cmd*)

Add a command to the end of the list of commands returned by the CmdGenerator.

Parameters
cmd (*str*) – Command to add to the list of commands to return.

class synapse.tests.utils.**DeprModule**(*core*, *conf=None*)

Bases: `CoreModule`

getModelDefs()

class synapse.tests.utils.**HttpReflector**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: `Handler`

Test handler which reflects get/post data back to the caller async **get**()

async **head**()

async **post**()

class synapse.tests.utils.**LibTst**(*runt*, *name=()*)

Bases: *Lib*

LibTst for testing!

addLibFuncs()

async **beep**(*valu*)

Example storm func

async **someargs**(*valu*, *bar=True*, *faz=None*)

Example storm func

class synapse.tests.utils.**PickleableMagicMock**(**args*, ***kw*)

Bases: MagicMock

class synapse.tests.utils.**ReloadCell**

Bases: *Cell*

async **addTestBadReload**()

async **addTestReload**()

async **getCellInfo**()

Return metadata specific for the Cell.

Notes

By default, this function returns information about the base Cell implementation, which reflects the base information in the Synapse Cell.

It is expected that implementers override the following Class attributes in order to provide meaningful version information:

`COMMIT` - A Git Commit `VERSION` - A Version tuple. `VERSTRING` - A Version string.

Returns

A Dictionary of metadata.

Return type

Dict

async **postAnit**()

Method called after self.__anit__() has completed, but before anit() returns the object to the caller.

class synapse.tests.utils.**StormPkgTest**(**args*, ***kwargs*)

Bases: *SynTest*

assetdir = None

`getTestCore(conf=None, dirn=None, prepkghook=None)`

Get a simple test Cortex as an async context manager.

Returns
A Cortex object.

Return type
s_cortex.Cortex

`async initTestCore(core)`

This is executed after the package has been loaded and a VCR context has been entered.

`pkgprotos = ()`

`vcr = None`

`class synapse.tests.utils.StreamEvent(*args, **kwargs)`

Bases: `StringIO`, `Event`

A combination of a io.StringIO object and a threading.Event object.

`setMesg(mesg)`

Clear the internal event and set a new message that is used to set the event.

Parameters
mesg (*str*) – The string to monitor for.

Returns
None

`write(s)`

Write string to file.

Returns the number of characters written, which is always equal to the length of the string.

`class synapse.tests.utils.SynTest(*args, **kwargs)`

Bases: `TestCase`

Mark all async test methods as s_glob.synchelp decorated.

Note: This precludes running a single unit test via path using the unittest module.

`async addCreatorDeleterRoles(core)`

Add two roles to a Cortex *proxy*, the *creator* and *deleter* roles. Creator allows for node:add, prop:set and tag:add actions. Deleter allows for node:del, prop:del and tag:del actions.

Parameters
core – Auth enabled cortex.

`addSvcToAha(aha, svcname, ctor, conf=None, dirn=None, provinfo=None)`

Creates as service and provision it in a Aha network via the provisioning API.

This assumes the Aha cell has a provision:listen and aha:urls set.

Parameters

- **aha** (*s_aha.AhaCell*) – Aha cell.
- **svcname** (*str*) – Service name.
- **ctor** – Service class to add.

- **conf** (*dict*) – Optional service conf.
- **dirn** (*str*) – Optional directory.
- **provinfo** (*dict*)) – Optional provisioning info.

Notes

The config data for the cell is pushed into dirn/cell.yaml. The cells are created with the `ctor.anit()` function.

`async addSvcToCore`(*svc*, *core*, *svcname='svc'*)

Add a service to a Cortex using telepath over tcp.

`async agenlen`(*x*, *obj*, *msg=None*)

Assert that the async generator produces x items

`async agenraises`(*exc*, *gfunc*)

Helper to validate that an async generator will throw an exception.

Parameters

- **exc** – Exception class to catch
- **gfunc** – async Generator

`async asyncraises`(*exc*, *coro*)

`checkNode`(*node*, *expected*)

`async checkNodes`(*core*, *ndefs*)

`eq`(*x*, *y*, *msg=None*)

Assert X is equal to Y

`eqOrNan`(*x*, *y*, *msg=None*)

Assert X is equal to Y or they are both NaN (needed since NaN != NaN)

`eqish`(*x*, *y*, *places=6*, *msg=None*)

Assert X is equal to Y within places decimal places

`async execToolMain`(*func*, *argv*)

`extendOutpFromPatch`(*outp*, *patch*)

Extend an Outp with lines from a magicMock object from withCliPromptMock.

Parameters

- **outp** (`TstOutPut`) – The outp to extend.
- **patch** (*mock.MagicMock*) – The patch object.

Returns

Returns none.

Return type

None

`false`(*x*, *msg=None*)

Assert X is False

ge(*x*, *y*, *msg=None*)

Assert that X is greater than or equal to Y

genraises(*exc*, *gfunc*, **args*, ***kwargs*)

Helper to validate that a generator function will throw an exception.

Parameters

- **exc** – Exception class to catch
- **gfunc** – Generator function to call.
- ***args** – Args passed to the generator function.
- ****kwargs** – Kwargs passed to the generator function.

Notes

Wrap a generator function in a list() call and execute that in a bound local using `self.raises(exc, boundlocal)`. The `list()` will consume the generator until complete or an exception occurs.

getAsyncLoggerStream(*logname*, *mesg=''*)

Async version of getLoggerStream.

Parameters

- **logname** (*str*) – Name of the logger to get.
- **mesg** (*str*) – A string which, if provided, sets the StreamEvent event if a message containing the string is written to the log.

Notes

The event object mixed in for the AsyncStreamEvent is a asyncio.Event object. This requires the user to await the Event specific calls as neccesary.

Examples

Do an action and wait for a specific log message to be written:

```
with self.getAsyncLoggerStream('synapse.foo.bar',
                               'big badda boom happened') as stream:
    # Do something that triggers a log message
    await doSomething()
    # Wait for the mesg to be written to the stream
    await stream.wait(timeout=10)

stream.seek(0)
mesgs = stream.read()
# Do something with messages
```

Returns
An AsyncStreamEvent object.

Return type
AsyncStreamEvent

getHttpSess(*auth=None*, *port=None*)

Get an aiohttp ClientSession with a CookieJar.

Parameters

- **auth** (*str, str*) – A tuple of username and password information for http auth.
- **port** (*int*) – Port number to connect to.

Notes

If auth and port are provided, the session will login to a Synapse cell hosted at localhost:port.

Returns
An aiohttp.ClientSession object.

Return type
aiohttp.ClientSession

getLoggerStream(*logname*, *mesg=''*)

Get a logger and attach a io.StringIO object to the logger to capture log messages.

Parameters

- **logname** (*str*) – Name of the logger to get.
- **mesg** (*str*) – A string which, if provided, sets the StreamEvent event if a message
- **log.** (*containing the string is written to the*) –

Examples

Do an action and get the stream of log messages to check against:

```
with self.getLoggerStream('synapse.foo.bar') as stream:
    # Do something that triggers a log message
    doSomething()

stream.seek(0)
mesgs = stream.read()
# Do something with messages
```

Do an action and wait for a specific log message to be written:

```
with self.getLoggerStream('synapse.foo.bar', 'big badda boom happened') as␣
↪stream:
    # Do something that triggers a log message
    doSomething()
    stream.wait(timeout=10)  # Wait for the mesg to be written to the stream

stream.seek(0)
mesgs = stream.read()
# Do something with messages
```

You can also reset the message and wait for another message to occur:

```
with self.getLoggerStream('synapse.foo.bar', 'big badda boom happened') as
↪stream:
    # Do something that triggers a log message
    doSomething()
    stream.wait(timeout=10)
    stream.setMesg('yo dawg')  # This will now wait for the 'yo dawg' string to
↪be written.
    stream.wait(timeout=10)

stream.seek(0)
mesgs = stream.read()
# Do something with messages
```

Notes

This **only** captures logs for the current process.

Yields
StreamEvent – A StreamEvent object

getMagicPromptColors(*patch*)

Get the colored lines from a MagicMock object from withCliPromptMock.

Parameters
patch (*mock.MagicMock*) – The MagicMock object from withCliPromptMock.

Returns
A list of tuples, containing color and line data.

Return type
list

getMagicPromptLines(*patch*)

Get the text lines from a MagicMock object from withCliPromptMock.

Parameters
patch (*mock.MagicMock*) – The MagicMock object from withCliPromptMock.

Returns
A list of lines.

Return type
list

getRegrAxon(*vers*, *conf=None*)

getRegrCore(*vers*, *conf=None*)

getRegrDir(**path*)

getStructuredAsyncLoggerStream(*logname*, *mesg=''*)

Async version of getLoggerStream which uses structured logging.

Parameters

- **logname** (*str*) – Name of the logger to get.
- **mesg** (*str*) – A string which, if provided, sets the StreamEvent event if a message containing the string is written to the log.

Notes

The event object mixed in for the AsyncStreamEvent is a asyncio.Event object. This requires the user to await the Event specific calls as needed. The messages written to the stream will be JSON lines.

Examples

Do an action and wait for a specific log message to be written:

```
with self.getStructuredAsyncLoggerStream('synapse.foo.bar',
                                         '"some JSON string"') as stream:
    # Do something that triggers a log message
    await doSomething()
    # Wait for the mesg to be written to the stream
    await stream.wait(timeout=10)

data = stream.getvalue()
raw_mesgs = [m for m in data.split('\n') if m]
msgs = [json.loads(m) for m in raw_mesgs]
# Do something with messages
```

Returns
An AsyncStreamEvent object.

Return type
AsyncStreamEvent

getTestAha(*conf=None*, *dirn=None*)

getTestAhaProv(*conf=None*, *dirn=None*)

Get an Aha cell that is configured for provisioning on aha.loop.vertex.link.

Parameters

- **conf** – Optional configuration information for the Aha cell.
- **dirn** – Optional path to create the Aha cell in.

Returns
The provisioned Aha cell.

Return type
s_aha.AhaCell

getTestAxon(*dirn=None*, *conf=None*)

Get a test Axon as an async context manager.

Returns
A Axon object.

Return type
s_axon.Axon

getTestCell(*ctor*, *conf=None*, *dirn=None*)

Get a test Cell.

`getTestCertDir`(*dirn*)

Patch the synapse.lib.certdir.certdir singleton and supporting functions with a CertDir instance backed by the provided directory.

Parameters
`dirn` (*str*) – The directory used to back the new CertDir singleton.

Returns
The patched CertDir object that is the current singleton.

Return type
s_certdir.CertDir

`getTestConfDir`(*name*, *conf=None*)

`getTestCore`(*conf=None*, *dirn=None*)

Get a simple test Cortex as an async context manager.

Returns
A Cortex object.

Return type
s_cortex.Cortex

`getTestCoreAndProxy`(*conf=None*, *dirn=None*)

Get a test Cortex and the Telepath Proxy to it.

Returns
The Cortex and a Proxy representing a CoreApi object.

Return type
(s_cortex.Cortex, s_cortex.CoreApi)

`getTestCoreProxSvc`(*ssvc*, *ssvc_conf=None*, *core_conf=None*)

Get a test Cortex, the Telepath Proxy to it, and a test service instance.

Parameters

- **ssvc** – Ctor to the Test Service.
- **ssvc_conf** – Service configuration.
- **core_conf** – Cortex configuration.

Returns
The Cortex, Proxy, and service instance.

Return type
(s_cortex.Cortex, s_cortex.CoreApi, testsvc)

`getTestCryo`(*dirn=None*, *conf=None*)

Get a simple test Cryocell as an async context manager.

Returns
Test cryocell.

Return type
s_cryotank.CryoCell

`getTestCryoAndProxy`(*dirn=None*)

Get a test Cryocell and the Telepath Proxy to it.

Returns
CryoCell, s_cryotank.CryoApi): The CryoCell and a Proxy representing a CryoApi object.

Return type
(s_cryotank

getTestDir(*mirror=None*, *copyfrom=None*, *chdir=False*, *startdir=None*)

Get a temporary directory for test purposes. This destroys the directory afterwards.

Parameters

- **mirror** (*str*) – A Synapse test directory to mirror into the test directory.
- **copyfrom** (*str*) – An arbitrary directory to copy into the test directory.
- **chdir** (*boolean*) – If true, chdir the current process to that directory. This is undone when the context manager exits.
- **startdir** (*str*) – The directory under which to place the temporary directory

Notes

The mirror argument is normally used to mirror test directory under `synapse/tests/files`. This is accomplished by passing in the name of the directory (such as `testcore`) as the mirror argument.

If the `mirror` argument is an absolute directory, that directory will be copied to the test directory.

Returns
The path to a temporary directory.

Return type
str

getTestDmon()

getTestFilePath(**names*)

getTestHive()

getTestHiveDmon()

getTestHiveFromDirn(*dirn*)

getTestJsonStor(*dirn=None*, *conf=None*)

getTestOutp()

Get a Output instance with a expects() function.

Returns
A TstOutPut instance.

Return type
TstOutPut

getTestProxy(*dmon*, *name*, ***kwargs*)

getTestReadWriteCores(*conf=None*, *dirn=None*)

Get a read/write core pair.

Notes

By default, this returns the same cortex. It is expected that a test which needs two distinct Cortexes implements the bridge themselves.

Returns
A tuple of Cortex objects.

Return type
(s_cortex.Cortex, s_cortex.Cortex)

`getTestSynDir()`
Combines getTestDir() and setSynDir() into one.

`getTestTeleHive()`

`getTestUrl`(*dmon*, *name*, ***opts*)

`gt`(*x*, *y*, *msg=None*)
Assert that X is greater than Y

`isin`(*member*, *container*, *msg=None*)
Assert a member is inside of a container.

`isinstance`(*obj*, *cls*, *msg=None*)
Assert a object is the instance of a given class or tuple of classes.

`istufo`(*obj*)
Check to see if an object is a tufo.

Parameters
obj (*object*) – Object being inspected.

Notes

This does not make any assumptions about the contents of the dictionary. This validates the object to be a tuple of length two, containing a str or None as the first value, and a dict as the second value.

Returns
None

`le`(*x*, *y*, *msg=None*)
Assert that X is less than or equal to Y

`len`(*x*, *obj*, *msg=None*)
Assert that the length of an object is equal to X

`lt`(*x*, *y*, *msg=None*)
Assert that X is less than Y

`ne`(*x*, *y*)
Assert X is not equal to Y

`nn`(*x*, *msg=None*)
Assert X is not None

`none`(*x*, *msg=None*)
Assert X is None

noprop(*info*, *prop*)

Assert a property is not present in a dictionary.

notin(*member*, *container*, *msg=None*)

Assert a member is not inside of a container.

printed(*msgs*, *text*)

raises(**args*, ***kwargs*)

Assert a function raises an exception.

redirectStdin(*new_stdin*)

Temporary replace stdin.

Parameters
`new_stdin` (*file-like object*) – file-like object.

Examples

Patch stdin with a string buffer:

```
inp = io.StringIO('stdin stuff\nanother line\n')
with self.redirectStdin(inp):
    main()
```

Here's a way to use this for code that's expecting the stdin buffer to have bytes:

```
inp = Mock()
inp.buffer = io.BytesIO(b'input data')
with self.redirectStdin(inp):
    main()
```

Returns
None

async **runCoreNodes**(*core*, *query*, *opts=None*)

Run a storm query through a Cortex as a SchedCoro and return the results.

setSynDir(*dirn*)

Sets s_common.syndir to a specific directory and then unsets it afterwards.

Parameters
`dirn` (*str*) – Directory to set syndir to.

Notes

This is to be used as a context manager.

setTstEnvars(***props*)

Set Environment variables for the purposes of running a specific test.

Parameters

- `**props` – A kwarg list of envars to set. The values set are run
- `strings.` (*through str() to ensure we're setting*) –

Examples

Run a test while a envar is set:

```
with self.setEnvars(magic='haha') as nop:
    ret = dostuff()
    self.true(ret)
```

Notes

This helper explicitly sets and unsets values in os.environ, as os.putenv does not automatically updates the os.environ object.

Yields
None. This context manager yields None. Upon exiting, envars are either removed from os.environ or reset to their previous values.

skip(*mesg*)

skipIfNexusReplay()
Allow skipping a test if SYNDEV_NEXUS_REPLAY envar is set.

Raises
unittest.SkipTest if SYNDEV_NEXUS_REPLAY envar is set to true value. –

skipIfNoInternet()
Allow skipping a test if SYN_TEST_SKIP_INTERNET envar is set.

Raises
unittest.SkipTest if SYN_TEST_SKIP_INTERNET envar is set to a integer greater than 1. –

skipIfNoPath(*path*, *mesg=None*)
Allows skipping a test if the test/files path does not exist.

Parameters

- **path** (*str*) – Path to check.
- **mesg** (*str*) – Optional additional message.

Raises
unittest.SkipTest if the path does not exist. –

skipLongTest()
Allow skipping a test if SYN_TEST_SKIP_LONG envar is set.

Raises
unittest.SkipTest if SYN_TEST_SKIP_LONG envar is set to a integer greater than 1. –

sorteq(*x*, *y*, *msg=None*)
Assert two sorted sequences are the same.

stablebuid(*valu=None*)
A stable buid generation for testing purposes

stableguid(*valu=None*)
A stable guid generation for testing purposes

`stormHasNoErr`(*mesgs*)

Raise an AssertionError if there is a message of type "err" in the list.

Parameters
mesgs (*list*) – A list of storm messages.

`stormHasNoWarnErr`(*mesgs*)

Raise an AssertionError if there is a message of type "err" or "warn" in the list.

Parameters
mesgs (*list*) – A list of storm messages.

`stormIsInErr`(*mesg*, *mesgs*)

Check if a string is present in all of the error messages from a stream of storm messages.

Parameters

- **mesg** (*str*) – A string to check.
- **mesgs** (*list*) – A list of storm messages.

`stormIsInPrint`(*mesg*, *mesgs*, *deguid=False*)

Check if a string is present in all of the print messages from a stream of storm messages.

Parameters

- **mesg** (*str*) – A string to check.
- **mesgs** (*list*) – A list of storm messages.

`stormIsInWarn`(*mesg*, *mesgs*)

Check if a string is present in all of the warn messages from a stream of storm messages.

Parameters

- **mesg** (*str*) – A string to check.
- **mesgs** (*list*) – A list of storm messages.

`stormNotInPrint`(*mesg*, *mesgs*)

Assert a string is not present in all of the print messages from a stream of storm messages.

Parameters

- **mesg** (*str*) – A string to check.
- **mesgs** (*list*) – A list of storm messages.

`stormNotInWarn`(*mesg*, *mesgs*)

Assert a string is not present in all of the warn messages from a stream of storm messages.

Parameters

- **mesg** (*str*) – A string to check.
- **mesgs** (*list*) – A list of storm messages.

`thisHostMust`(***props*)

Requires a host having a specific property.

Parameters
****props** –

Raises
unittest.SkipTest if the required property is missing. –

thisHostMustNot(***props*)

Requires a host to not have a specific property.

Parameters
****props** –

Raises
unittest.SkipTest if the required property is missing. –

true(*x*, *msg=None*)

Assert X is True

withCliPromptMock()

Context manager to mock our use of Prompt Toolkit's print_formatted_text function.

Returns
Yields a mock.MagikMock object.

Return type
mock.MagicMock

withCliPromptMockExtendOutp(*outp*)

Context manager to mock our use of Prompt Toolkit's print_formatted_text function and extend the lines to an an output object.

Parameters
outp (TstOutPut) – The outp to extend.

Notes

This extends the outp with the lines AFTER the context manager has exited.

Returns
Yields a mock.MagicMock object.

Return type
mock.MagicMock

withNexusReplay(*replay=False*)

Patch so that the Nexus apply log is applied twice. Useful to verify idempotency.

Parameters
replay (*bool*) – Set the default value of resolving the existence of SYNDEV_NEXUS_REPLAY variable. This can be used to force the apply patch without using the environment variable.

Notes

This is applied if the environment variable SYNDEV_NEXUS_REPLAY is set to a non zero value or the replay argument is set to True.

Returns
An exitstack object.

Return type
contextlib.ExitStack

withSetLoggingMock()

Context manager to mock calls to the setlogging function to avoid unittests calling logging.basicconfig.

Returns
Yields a mock.MagicMock object.

Return type
mock.MagicMock

withStableUids()

A context manager that generates guids and buids in sequence so that successive test runs use the same data

withTestCmdr(*cmdg*)

class synapse.tests.utils.**TestCmd**(*runt*, *runtsafe*)

Bases: *Cmd*

A test command

async **execStormCmd**(*runt*, *genr*)

Abstract base method

forms = {'input': ['test:str', 'inet:ipv6'], 'nodedata': [('foo', 'inet:ipv4'), ('bar', 'inet:fqdn')], 'output': ['inet:fqdn']}

getArgParser()

name = 'testcmd'

class synapse.tests.utils.**TestModule**(*core*, *conf=None*)

Bases: *CoreModule*

async **addTestRecords**(*snap*, *items*)

getModelDefs()

getStormCmds()

Module implementers may override this to provide a list of Storm commands which will be loaded into the Cortex.

Returns
A list of Storm Command classes (not instances).

Return type
list

async **initCoreModule**()

Module implementers may override this method to initialize the module after the Cortex has completed and is accessible to perform storage operations.

Notes

This is the preferred function to override for implementing custom code that needs to be executed during Cortex startup.

Any exception raised within this method will remove the module from the list of currently loaded modules.

This is called for modules after getModelDefs() and getStormCmds() has been called, in order to allow for model loading and storm command loading prior to code execution offered by initCoreModule.

A failure during initCoreModule will not unload data model or storm commands registered by the module.

Returns
None

`testguid = '8f1401de15918358d5247e21ca29a814'`

`class synapse.tests.utils.TestRunt(name, **kwargs)`

Bases: `object`

`getStorNode(form)`

`class synapse.tests.utils.TestSubType(modl, name, info, opts)`

Bases: *Type*

`norm(valu)`

Normalize the value for a given type.

Parameters
`valu` (*obj*) – The value to normalize.

Returns
The normalized valu, info tuple.

Return type
((obj,dict))

Notes

The info dictionary uses the following key conventions:
subs (dict): The normalized sub-fields as name: valu entries.

`repr(norm)`

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

`stortype: int = 4`

`class synapse.tests.utils.TestType(modl, name, info, opts)`

Bases: *Type*

`postTypeInit()`

`stortype: int = 1`

`class synapse.tests.utils.ThreeType(modl, name, info, opts)`

Bases: *Type*

norm(*valu*)

Normalize the value for a given type.

Parameters
valu (*obj*) – The value to normalize.

Returns
The normalized valu, info tuple.

Return type
((obj,dict))

Notes

The info dictionary uses the following key conventions:
subs (dict): The normalized sub-fields as name: valu entries.

repr(*valu*)

Return a printable representation for the value. This may return a string or a tuple of values for display purposes.

stortype: int = 2

class synapse.tests.utils.**TstEnv**

Bases: object

add(*name*, *item*, *fini=False*)

async **fini**()

class synapse.tests.utils.**TstOutPut**

Bases: *OutPutStr*

clear()

expect(*substr*, *throw=True*)

Check if a string is present in the messages captured by the OutPutStr object.

Parameters

- **substr** (*str*) – String to check for the existence of.
- **throw** (*bool*) – If True, a missing substr results in a Exception being thrown.

Returns
True if the string is present; False if the string is not present and throw is False.

Return type
bool

async synapse.tests.utils.**alist**(*coro*)

synapse.tests.utils.**deguidify**(*x*)

synapse.tests.utils.**matchContexts**(*testself*)

synapse.tests.utils.**norm**(*z*)

synapse.tools package

Subpackages

synapse.tools.aha package

Subpackages

synapse.tools.aha.provision package

Submodules

synapse.tools.aha.provision.service module

async `synapse.tools.aha.provision.service.`**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.aha.provision.user module

async `synapse.tools.aha.provision.user.`**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

Submodules

synapse.tools.aha.easycert module

`synapse.tools.aha.easycert.`**getArgParser**()

async `synapse.tools.aha.easycert.`**main**(*argv*, *outp=None*)

synapse.tools.aha.enroll module

async `synapse.tools.aha.enroll.`**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.aha.list module

async `synapse.tools.aha.list.`**main**(*argv*, *outp=None*)

synapse.tools.cryo package

Submodules

synapse.tools.cryo.cat module

async `synapse.tools.cryo.cat.`**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.cryo.list module

async synapse.tools.cryo.list.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.docker package

Submodules

synapse.tools.docker.validate module

synapse.tools.docker.validate.**checkCRL**(*outp*, *sigd*, *certdir*)

synapse.tools.docker.validate.**checkCosign**(*outp*)

synapse.tools.docker.validate.**checkCosignSignature**(*outp*, *pubk_byts*, *image_to_verify*)

synapse.tools.docker.validate.**getArgParser**()

synapse.tools.docker.validate.**getCosignSignature**(*outp*, *image*)

synapse.tools.docker.validate.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.hive package

Submodules

synapse.tools.hive.load module

async synapse.tools.hive.load.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.hive.save module

async synapse.tools.hive.save.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

Submodules

synapse.tools.autodoc module

class synapse.tools.autodoc.**DocHelp**(*ctors*, *types*, *forms*, *props*, *univs*)

Bases: object

Helper to pre-compute all doc strings hierarchically

async synapse.tools.autodoc.**docConfdefs**(*ctor*)

async synapse.tools.autodoc.**docModel**(*outp*, *core*)

async synapse.tools.autodoc.**docStormTypes**()

async synapse.tools.autodoc.**docStormpkg**(*pkgpath*)

async synapse.tools.autodoc.**docStormsvc**(*ctor*)

synapse.tools.autodoc.**has_popts_data**(*props*)

synapse.tools.autodoc.**lookupedgesforform**(*form: str*, *edges: List[Tuple[Tuple[str | None, str, str | None], Dict[str, str]]]*) → Dict[str, List[Tuple[Tuple[str | None, str, str | None], Dict[str, str]]]]

async synapse.tools.autodoc.**main**(*argv*, *outp=None*)

synapse.tools.autodoc.**makeargparser**()

synapse.tools.autodoc.**processCtors**(*rst*, *dochelp*, *ctors*)

Parameters

- **rst** (RstHelp) –
- **dochelp** (DocHelp) –
- **ctors** (*list*) –

Returns
None

synapse.tools.autodoc.**processFormsProps**(*rst*, *dochelp*, *forms*, *univ_names*, *alledges*)

async synapse.tools.autodoc.**processStormCmds**(*rst*, *pkgname*, *commands*)

Parameters

- **rst** (RstHelp) –
- **pkgname** (*str*) –
- **commands** (*list*) –

Returns
None

async synapse.tools.autodoc.**processStormModules**(*rst*, *pkgname*, *modules*)

synapse.tools.autodoc.**processTypes**(*rst*, *dochelp*, *types*)

Parameters

- **rst** (RstHelp) –
- **dochelp** (DocHelp) –
- **ctors** (*list*) –

Returns
None

synapse.tools.autodoc.**processUnivs**(*rst*, *dochelp*, *univs*)

synapse.tools.axon2axon module

async synapse.tools.axon2axon.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.backup module

synapse.tools.backup.**backup**(*srcdir*, *dstdir*, *skipdirs=None*)

Create a backup of a Synapse application.

Parameters

- **srcdir** (*str*) – Path to the directory to backup.
- **dstdir** (*str*) – Path to backup target directory.
- **skipdirs** (*list or None*) – Optional list of relative directory name glob patterns to exclude from the backup.

Note: Running this method from the same process as a running user of the directory may lead to a segmentation fault

synapse.tools.backup.**backup_lmdb**(*env*, *dstdir*, *txn=None*)

synapse.tools.backup.**capturelmdbs**(*srcdir*, *skipdirs=None*, *onlydirs=None*)

A context manager that opens all the lmdb files under a srcdir and makes a read transaction. All transactions are aborted and environments closed when the context is exited.

Yields

Dict[str, Tuple[lmdb.Environment, lmdb.Transaction]] – Maps path to environment, transaction

synapse.tools.backup.**main**(*argv*)

synapse.tools.backup.**parse_args**(*argv*)

synapse.tools.backup.**txnbackup**(*lmdbinfo*, *srcdir*, *dstdir*, *skipdirs=None*)

Create a backup of a Synapse application under a (hopefully consistent) set of transactions.

Parameters

- **lmdbinfo** (*Dict[str, Tuple[lmdb.Environment, lmdb.Transaction]]*) – Maps of path to environment, transaction
- **srcdir** (*str*) – Path to the directory to backup.
- **dstdir** (*str*) – Path to backup target directory.
- **skipdirs** (*list or None*) – Optional list of relative directory name glob patterns to exclude from the backup.

Note: Running this method from the same process as a running user of the directory may lead to a segmentation fault

synapse.tools.cellauth module

async synapse.tools.cellauth.**handleList**(*opts*)

async synapse.tools.cellauth.**handleModify**(*opts*)

async synapse.tools.cellauth.**main**(*argv*, *outprint=None*)

synapse.tools.cellauth.**makeargparser**()

async synapse.tools.cellauth.**printuser**(*user*, *details=False*, *cell=None*)

synapse.tools.cellauth.**reprrule**(*rule*)

synapse.tools.cmdr module

async synapse.tools.cmdr.**main**(*argv*)

async synapse.tools.cmdr.**runcmdr**(*argv*, *item*)

synapse.tools.csvtool module

async synapse.tools.csvtool.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.csvtool.**makeargparser**(*outp*)

async synapse.tools.csvtool.**runCsvExport**(*opts*, *outp*, *text*, *stormopts*)

async synapse.tools.csvtool.**runCsvImport**(*opts*, *outp*, *text*, *stormopts*)

synapse.tools.easycert module

synapse.tools.easycert.**main**(*argv*, *outp=None*)

synapse.tools.feed module

async synapse.tools.feed.**addFeedData**(*core*, *outp*, *feedformat*, *debug=False*, **paths*, *chunksize=1000*, *offset=0*, *viewiden=None*)

synapse.tools.feed.**getItems**(**paths*)

async synapse.tools.feed.**main**(*argv*, *outp=None*)

synapse.tools.feed.**makeargparser**()

synapse.tools.genpkg module

synapse.tools.genpkg.**getStormStr**(*fn*)

synapse.tools.genpkg.**loadOpticFiles**(*pkgdef*, *path*)

synapse.tools.genpkg.**loadOpticWorkflows**(*pkgdef*, *path*)

synapse.tools.genpkg.**loadPkgProto**(*path*, *opticdir=None*, *no_docs=False*, *readonly=False*)

Get a Storm Package definition from disk.

Parameters

- **path** (*str*) – Path to the package .yaml file on disk.
- **opticdir** (*str*) – Path to optional Optic module code to add to the Storm Package.
- **no_docs** (*bool*) – If true, omit inline documentation content if it is not present on disk.
- **readonly** (*bool*) – If set, open files in read-only mode. If files are missing, that will raise a NoSuchFile exception.

Returns
A Storm package definition.

Return type
dict

async synapse.tools.genpkg.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.genpkg.**tryLoadPkgProto**(*fp*, *opticdir=None*, *readonly=False*)

Try to get a Storm Package prototype from disk with or without inline documentation.

Parameters

- **fp** (*str*) – Path to the package .yaml file on disk.
- **opticdir** (*str*) – Path to optional Optic module code to add to the Storm Package.
- **readonly** (*bool*) – If set, open files in read-only mode. If files are missing, that will raise a NoSuchFile exception.

Returns
A Storm package definition.

Return type
dict

synapse.tools.guid module

synapse.tools.guid.**main**(*argv*, *outp=None*)

synapse.tools.healthcheck module

synapse.tools.healthcheck.**format_component**(*e*, *mesg: str*) → dict

async synapse.tools.healthcheck.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.healthcheck.**makeargparser**()

synapse.tools.healthcheck.**serialize**(*ret*)

synapse.tools.json2mpk module

synapse.tools.json2mpk.**getArgParser**()

synapse.tools.json2mpk.**main**(*argv*, *outp=None*)

synapse.tools.livebackup module

async synapse.tools.livebackup.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.modrole module

async synapse.tools.modrole.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.modrole.**printrole**(*role*, *outp*)

synapse.tools.moduser module

async synapse.tools.moduser.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.moduser.**printuser**(*user*, *outp*)

synapse.tools.promote module

async synapse.tools.promote.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.pullfile module

async synapse.tools.pullfile.**main**(*argv*, *outp=None*)

synapse.tools.pullfile.**setup**()

synapse.tools.pushfile module

async synapse.tools.pushfile.**main**(*argv*, *outp=None*)

synapse.tools.pushfile.**makeargparser**()

synapse.tools.reload module

synapse.tools.reload.**getArgParser**()

async synapse.tools.reload.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.rstorm module

async synapse.tools.rstorm.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.tools.storm module

class synapse.tools.storm.**ExportCmd**(*cli*, ***opts*)

Bases: *StormCliCmd*

Export the results of a storm query into a nodes file.

Example

// Export nodes to a file !export dnsa.nodes { inet:fqdn#mynodes -> inet:dns:a }

// Export nodes to a file and only include specific tags !export fqdn.nodes { inet:fqdn#mynodes } –include-tags footag

getArgParser()

async **runCmdOpts**(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters

opts (*dict*) – Options dictionary.

class synapse.tools.storm.**HelpCmd**(*cli*, ***opts*)

Bases: *CmdHelp*

List interpreter extended commands and display help output.

Example

!help foocmd

class synapse.tools.storm.**PullFileCmd**(*cli*, ***opts*)

Bases: *StormCliCmd*

Download a file by sha256 and store it locally.

Example

!pullfile c00adfcc316f8b00772cdbce2505b9ea539d74f42861801eceb1017a44344ed3 /path/to/savefile

getArgParser()

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.tools.storm.**PushFileCmd**(*cli*, ***opts*)

Bases: *StormCliCmd*

Upload a file and create a file:bytes node.

Example

!pushfile /path/to/file

getArgParser()

async runCmdOpts(*opts*)

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
opts (*dict*) – Options dictionary.

class synapse.tools.storm.**QuitCmd**(*cli*, ***opts*)

Bases: *CmdQuit*

Quit the current command line interpreter.

Example

!quit

class synapse.tools.storm.**RunFileCmd**(*cli*, ***opts*)

Bases: *StormCliCmd*

Run a local storm file.

Example

!runfile /path/to/file.storm

`getArgParser()`

`async runCmdOpts(opts)`

Perform the command actions. Must be implemented by Cmd implementers.

Parameters
`opts` (*dict*) – Options dictionary.

`class synapse.tools.storm.StormCli`

Bases: *Cli*

`async handleErr(mesg)`

`histfile = 'storm_history'`

`initCmdClasses()`

`printf(mesg, addnl=True, color=None)`

`async runCmdLine(line, opts=None)`

Run a single command line.

Parameters
`line` (*str*) – Line to execute.

Examples

Execute the 'woot' command with the 'help' switch:

await cli.runCmdLine('woot –help')

Returns
Arbitrary data from the cmd class.

Return type
object

`async storm(text, opts=None)`

`class synapse.tools.storm.StormCliCmd(cli, **opts)`

Bases: *Cmd*

`getArgParser()`

`getCmdOpts(text)`

Use the _cmd_syntax def to split/parse/normalize the cmd line.

Parameters
`text` (*str*) – Command to process.

Notes

This is implemented independent of argparse (et al) due to the need for syntax aware argument splitting. Also, allows different split per command type

Returns
An opts dictionary.

Return type
dict

class synapse.tools.storm.**StormCompleter**(*cli*)

Bases: `Completer`

get_completions(*document*, *complete_event*)

This should be a generator that yields `Completion` instances.

If the generation of completions is something expensive (that takes a lot of time), consider wrapping this *Completer* class in a *ThreadedCompleter*. In that case, the completer algorithm runs in a background thread and completions will be displayed as soon as they arrive.

Parameters

- **document** – `Document` instance.
- **complete_event** – `CompleteEvent` instance.

async **get_completions_async**(*document*, *complete_event*)

Asynchronous generator for completions. (Probably, you won't have to override this.)

Asynchronous generator of `Completion` objects.

async **load**()

synapse.tools.storm.**cmplgenr**(**genrs*, *prefix=''*)

Iterate over all the generators/iterators passed in as args and return Completions from them. If prefix is specified, make sure the current item in the generator starts with the prefix value.

synapse.tools.storm.**getArgParser**()

async synapse.tools.storm.**main**(*argv*, *outp=<synapse.lib.output.OutPut object>*)

synapse.utils package

Subpackages

synapse.utils.stormcov package

synapse.utils.stormcov.**coverage_init**(*reg*, *options*)

Submodules

synapse.utils.stormcov.plugin module

class `synapse.utils.stormcov.plugin.`**`PivotTracer`**(*parent*)

Bases: `FileTracer`

`PARSE_METHODS = {'getNodeByNdef', 'nodesByPropArray', 'nodesByPropValu', 'nodesByTag'}`

`dynamic_source_filename`(*filename*, *frame*)

Get a dynamically computed source file name.

Some plug-ins need to compute the source file name dynamically for each frame.

This function will not be invoked if *`has_dynamic_source_filename()`* returns False.

Returns the source file name for this frame, or None if this frame shouldn't be measured.

`has_dynamic_source_filename`()

Does this FileTracer have dynamic source file names?

FileTracers can provide dynamically determined file names by implementing *`dynamic_source_filename()`*. Invoking that function is expensive. To determine whether to invoke it, coverage.py uses the result of this function to know if it needs to bother invoking *`dynamic_source_filename()`*.

See `CoveragePlugin.file_tracer()` for details about static and dynamic file names.

Returns True if *`dynamic_source_filename()`* should be called to get dynamic source file names.

`line_number_range`(*frame*)

Get the range of source line numbers for a given a call frame.

The call frame is examined, and the source line number in the original file is returned. The return value is a pair of numbers, the starting line number and the ending line number, both inclusive. For example, returning (5, 7) means that lines 5, 6, and 7 should be considered executed.

This function might decide that the frame doesn't indicate any lines from the source file were executed. Return (-1, -1) in this case to tell coverage.py that no lines should be recorded for this frame.

class `synapse.utils.stormcov.plugin.`**`StormCtrlTracer`**(*parent*)

Bases: `FileTracer`

`dynamic_source_filename`(*filename*, *frame*)

Get a dynamically computed source file name.

Some plug-ins need to compute the source file name dynamically for each frame.

This function will not be invoked if *`has_dynamic_source_filename()`* returns False.

Returns the source file name for this frame, or None if this frame shouldn't be measured.

`has_dynamic_source_filename`()

Does this FileTracer have dynamic source file names?

FileTracers can provide dynamically determined file names by implementing *`dynamic_source_filename()`*. Invoking that function is expensive. To determine whether to invoke it, coverage.py uses the result of this function to know if it needs to bother invoking *`dynamic_source_filename()`*.

See `CoveragePlugin.file_tracer()` for details about static and dynamic file names.

Returns True if `dynamic_source_filename()` should be called to get dynamic source file names.

line_number_range(*frame*)

Get the range of source line numbers for a given a call frame.

The call frame is examined, and the source line number in the original file is returned. The return value is a pair of numbers, the starting line number and the ending line number, both inclusive. For example, returning (5, 7) means that lines 5, 6, and 7 should be considered executed.

This function might decide that the frame doesn't indicate any lines from the source file were executed. Return (-1, -1) in this case to tell coverage.py that no lines should be recorded for this frame.

class synapse.utils.stormcov.plugin.**StormPlugin**(*options*)

Bases: `CoveragePlugin`, `FileTracer`

PARSE_METHODS = {'compute', 'getPivsIn', 'getPivsOut', 'lift', 'once'}

dynamic_source_filename(*filename*, *frame*, *force=False*)

Get a dynamically computed source file name.

Some plug-ins need to compute the source file name dynamically for each frame.

This function will not be invoked if `has_dynamic_source_filename()` returns False.

Returns the source file name for this frame, or None if this frame shouldn't be measured.

file_reporter(*filename*)

Get the `FileReporter` class to use for a file.

Plug-in type: file tracer.

This will only be invoked if *filename* returns non-None from `file_tracer()`. It's an error to return None from this method.

Returns a `FileReporter` object to use to report on *filename*, or the string *"python"* to have coverage.py treat the file as Python.

file_tracer(*filename*)

Get a `FileTracer` object for a file.

Plug-in type: file tracer.

Every Python source file is offered to your plug-in to give it a chance to take responsibility for tracing the file. If your plug-in can handle the file, it should return a `FileTracer` object. Otherwise return None.

There is no way to register your plug-in for particular files. Instead, this method is invoked for all files as they are executed, and the plug-in decides whether it can trace the file or not. Be prepared for *filename* to refer to all kinds of files that have nothing to do with your plug-in.

The file name will be a Python file being executed. There are two broad categories of behavior for a plug-in, depending on the kind of files your plug-in supports:

- Static file names: each of your original source files has been converted into a distinct Python file. Your plug-in is invoked with the Python file name, and it maps it back to its original source file.
- Dynamic file names: all of your source files are executed by the same Python file. In this case, your plug-in implements `FileTracer.dynamic_source_filename()` to provide the actual source file for each execution frame.

filename is a string, the path to the file being considered. This is the absolute real path to the file. If you are comparing to other paths, be sure to take this into account.

Returns a `FileTracer` object to use to trace *filename*, or None if this plug-in cannot trace this file.

find_executable_files(*src_dir*)

Yield all of the executable files in *src_dir*, recursively.

Plug-in type: file tracer.

Executability is a plug-in-specific property, but generally means files which would have been considered for coverage analysis, had they been included automatically.

Returns or yields a sequence of strings, the paths to files that could have been executed, including files that had been executed.

find_storm_files(*dirn*)

find_subqueries(*tree*, *path*)

has_dynamic_source_filename()

Does this FileTracer have dynamic source file names?

FileTracers can provide dynamically determined file names by implementing *dynamic_source_filename()*. Invoking that function is expensive. To determine whether to invoke it, coverage.py uses the result of this function to know if it needs to bother invoking *dynamic_source_filename()*.

See `CoveragePlugin.file_tracer()` for details about static and dynamic file names.

Returns True if *dynamic_source_filename()* should be called to get dynamic source file names.

line_number_range(*frame*)

Get the range of source line numbers for a given a call frame.

The call frame is examined, and the source line number in the original file is returned. The return value is a pair of numbers, the starting line number and the ending line number, both inclusive. For example, returning (5, 7) means that lines 5, 6, and 7 should be considered executed.

This function might decide that the frame doesn't indicate any lines from the source file were executed. Return (-1, -1) in this case to tell coverage.py that no lines should be recorded for this frame.

class synapse.utils.stormcov.plugin.**StormReporter**(*filename*, *parser*)

Bases: `FileReporter`

lines()

Get the executable lines in this file.

Your plug-in must determine which lines in the file were possibly executable. This method returns a set of those line numbers.

Returns a set of line numbers.

source()

Get the source for the file.

Returns a Unicode string.

The base implementation simply reads the *self.filename* file and decodes it as UTF-8. Override this method if your file isn't readable as a text file, or if you need other encoding support.

Submodules

synapse.utils.getrefs module

synapse.utils.getrefs.**download_refs**(*schema*)

synapse.utils.getrefs.**download_refs_handler**(*uri*)

This function downloads the JSON schema at the given URI, parses the given URI to get the path component, and then saves the referenced schema to the 'jsonschemas' directory of synapse.data.

synapse.utils.getrefs.**main**(*argv*)

synapse.utils.getrefs.**parse_args**(*argv*)

10.1.2 Submodules

10.1.3 synapse.axon module

class synapse.axon.**Axon**

Bases: *Cell*

byterange = False

cellapi

alias of *AxonApi*

confdefs = {'http:proxy': {'description': 'An aiohttp-socks compatible proxy URL to use in the wget API.', 'type': 'string'}, 'max:bytes': {'description': 'The maximum number of bytes that can be stored in the Axon.', 'hidecmdl': True, 'minimum': 1, 'type': 'integer'}, 'max:count': {'description': 'The maximum number of files that can be stored in the Axon.', 'hidecmdl': True, 'minimum': 1, 'type': 'integer'}, 'tls:ca:dir': {'description': 'An optional directory of CAs which are added to the TLS CA chain for wget and wput APIs.', 'type': 'string'}}

async **csvrows**(*sha256*, *dialect='excel'*, *errors='ignore'*, ***fmtparams*)

async **del_**(*sha256*)

Remove the given bytes from the Axon by sha256.

Parameters
sha256 (*bytes*) – The sha256, in bytes, to remove from the Axon.

Returns
True if the file is removed; false if the file is not present.

Return type
boolean

async **dels**(*sha256s*)

Given a list of sha256 hashes, delete the files from the Axon.

Parameters
sha256s (*list*) – A list of sha256 hashes in bytes form.

Returns
A list of booleans, indicating if the file was deleted or not.

Return type
list

async **get**(*sha256*, *offs=None*, *size=None*)

Get bytes of a file.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file in bytes.
- **offs** (*int*) – The offset to start reading from.
- **size** (*int*) – The total number of bytes to read.

Examples

Get the bytes from an Axon and process them:

```
buf = b''
async for bytz in axon.get(sha256):
    buf =+ bytz

await dostuff(buf)
```

Yields
bytes – Chunks of the file bytes.

Raises
synapse.exc.NoSuchFile – If the file does not exist.

async **getCellInfo**()

Return metadata specific for the Cell.

Notes

By default, this function returns information about the base Cell implementation, which reflects the base information in the Synapse Cell.

It is expected that implementers override the following Class attributes in order to provide meaningful version information:

`COMMIT` - A Git Commit `VERSION` - A Version tuple. `VERSTRING` - A Version string.

Returns
A Dictionary of metadata.

Return type
Dict

async **has**(*sha256*)

Check if the Axon has a file.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
True if the Axon has the file; false otherwise.

Return type
boolean

async hashes(*offs*, *wait=False*, *timeout=None*)

Yield hash rows for files that exist in the Axon in added order starting at an offset.

Parameters

- **offs** (*int*) – The index offset.
- **wait** (*boolean*) – Wait for new results and yield them in realtime.
- **timeout** (*int*) – Max time to wait for new results.

Yields
(int, (bytes, int)) – An index offset and the file SHA-256 and size.

Note: If the same hash was deleted and then added back, the same hash will be yielded twice.

async hashset(*sha256*)

Calculate additional hashes for a file in the Axon.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
A dictionary containing hashes of the file.

Return type
dict

async history(*tick*, *tock=None*)

Yield hash rows for files that existing in the Axon after a given point in time.

Parameters

- **tick** (*int*) – The starting time (in epoch milliseconds).
- **tock** (*int*) – The ending time to stop iterating at (in epoch milliseconds).

Yields
(int, (bytes, int)) – A tuple containing time of the hash was added and the file SHA-256 and size.

holdHashLock(*hashbyts*)

A context manager that synchronizes edit access to a blob.

Parameters
hashbyts (*bytes*) – The blob to hold the lock for.

async initServiceRuntime()

async initServiceStorage()

async iterMpkFile(*sha256*)

Yield items from a MsgPack (.mpk) file in the Axon.

Parameters
sha256 (*str*) – The sha256 hash of the file as a string.

Yields
Unpacked items from the bytes.

async **jsonlines**(*sha256*, *errors='ignore'*)

async **metrics**()

Get the runtime metrics of the Axon.

Returns

A dictionary of runtime data about the Axon.

Return type

dict

async **postfiles**(*fields*, *url*, *params=None*, *headers=None*, *method='POST'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

Send files from the axon as fields in a multipart/form-data HTTP request.

Parameters

- **fields** (*list*) – List of dicts containing the fields to add to the request as form-data.
- **url** (*str*) – The URL to retrieve.
- **params** (*dict*) – Additional parameters to add to the URL.
- **headers** (*dict*) – Additional HTTP headers to add in the request.
- **method** (*str*) – The HTTP method to use.
- **ssl** (*bool*) – Perform SSL verification.
- **timeout** (*int*) – The timeout of the request, in seconds.
- **proxy** (*bool | str | null*) – Use a specific proxy or disable proxy use.
- **ssl_opts** (*dict*) – Additional SSL/TLS options.

Notes

The dictionaries in the fields list may contain the following values:

```
{
    'name': <str> - Name of the field.
    'sha256': <str> - SHA256 hash of the file to submit for this field.
    'value': <str> - Value for the field. Ignored if a sha256 has been␣
↪specified.
    'filename': <str> - Optional filename for the field.
    'content_type': <str> - Optional content type for the field.
    'content_transfer_encoding': <str> - Optional content-transfer-encoding␣
↪header for the field.
}
```

The ssl_opts dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl␣
↪argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can␣
↪be included in client_cert.
}
```

The dictionary returned by this may contain the following values:

```
{
    'ok': <boolean> - False if there were exceptions retrieving the URL.
    'err': <tuple> - Tuple of the error type and information if an exception␣
↪occurred.
    'url': <str> - The URL retrieved (which could have been redirected)
    'code': <int> - The response code.
    'body': <bytes> - The response body.
    'reason': <str> - The reason phrase for the HTTP status code.
    'headers': <dict> - The response headers as a dictionary.
}
```

Returns
An information dictionary containing the results of the request.

Return type
dict

`async put(byts)`

Store bytes in the Axon.

Parameters
byts (*bytes*) – The bytes to store in the Axon.

Notes

This API should not be used for files greater than 128 MiB in size.

Returns
A tuple with the file size and sha256 hash of the bytes.

Return type
tuple(int, bytes)

`async puts(files)`

Store a set of bytes in the Axon.

Parameters
files (*list*) – A list of bytes to store in the Axon.

Notes

This API should not be used for storing more than 128 MiB of bytes at once.

Returns
A list containing tuples of file size and sha256 hash of the saved bytes.

Return type
list(tuple(int, bytes))

`async readlines(sha256, errors='ignore')`

`async save(sha256, genr, size)`

Save a generator of bytes to the Axon.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file in bytes.
- **genr** – The bytes generator function.

Returns
The size of the bytes saved.

Return type
int

async size(*sha256*)

Get the size of a file in the Axon.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
The size of the file, in bytes. If not present, None is returned.

Return type
int

async upload()

Get an Upload object.

Notes

The UpLoad object should be used to manage uploads greater than 128 MiB in size.

Examples

Use an UpLoad object to upload a file to the Axon:

```
async with await axon.upload() as upfd:
    # Assumes bytesGenerator yields bytes
    async for byts in bytsgenerator():
        await upfd.write(byts)
    await upfd.save()
```

Use a single UpLoad object to save multiple files:

```
async with await axon.upload() as upfd:
    for fp in file_paths:
        # Assumes bytesGenerator yields bytes
        async for byts in bytsgenerator(fp):
            await upfd.write(byts)
        await upfd.save()
```

Returns
An Upload manager object.

Return type
UpLoad

async wants(*sha256s*)

Get a list of sha256 values the axon does not have from a input list.

Parameters

sha256s (*list*) – A list of sha256 values as bytes.

Returns

A list of bytes containing the sha256 hashes the Axon does not have.

Return type

list

async wget(*url*, *params=None*, *headers=None*, *json=None*, *body=None*, *method='GET'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

Stream a file download directly into the Axon.

Parameters

- **url** (*str*) – The URL to retrieve.
- **params** (*dict*) – Additional parameters to add to the URL.
- **headers** (*dict*) – Additional HTTP headers to add in the request.
- **json** – A JSON body which is included with the request.
- **body** – The body to be included in the request.
- **method** (*str*) – The HTTP method to use.
- **ssl** (*bool*) – Perform SSL verification.
- **timeout** (*int*) – The timeout of the request, in seconds.
- **proxy** (*bool | str | null*) – Use a specific proxy or disable proxy use.
- **ssl_opts** (*dict*) – Additional SSL/TLS options.

Notes

The response body will be stored, regardless of the response code. The ok value in the response does not reflect that a status code, such as a 404, was encountered when retrieving the URL.

The ssl_opts dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl␣
↪argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can␣
↪be included in client_cert.
}
```

The dictionary returned by this may contain the following values:

```
{
    'ok': <boolean> - False if there were exceptions retrieving the URL.
    'url': <str> - The URL retrieved (which could have been redirected). This␣
↪is a url-decoded string.
    'code': <int> - The response code.
```

(continues on next page)

(continued from previous page)

```
    'reason': <str> - The reason phrase for the HTTP status code.
    'mesg': <str> - An error message if there was an exception when retrieving
↪the URL.
    'err': <tuple> - An error tuple if there was an exception when retrieving
↪the URL.
    'headers': <dict> - The response headers as a dictionary.
    'size': <int> - The size in bytes of the response body.
    'hashes': {
        'md5': <str> - The MD5 hash of the response body.
        'sha1': <str> - The SHA1 hash of the response body.
        'sha256': <str> - The SHA256 hash of the response body.
        'sha512': <str> - The SHA512 hash of the response body.
    },
    'request': {
        'url': The request URL. This is a url-decoded string.
        'headers': The request headers.
        'method': The request method.
    }
    'history': A sequence of response bodies to track any redirects, not
↪including hashes.
}
```

Returns
An information dictionary containing the results of the request.

Return type
dict

async wput(*sha256*, *url*, *params=None*, *headers=None*, *method='PUT'*, *ssl=True*, *timeout=None*, *filename=None*, *filemime=None*, *proxy=None*, *ssl_opts=None*)

Stream a blob from the axon as the body of an HTTP request.

class synapse.axon.**AxonApi**

Bases: *CellApi*, *Share*

async csvrows(*sha256*, *dialect='excel'*, *errors='ignore'*, ***fmtparams*)

Yield CSV rows from a CSV file.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file.
- **dialect** (*str*) – The CSV dialect to use.
- **errors** (*str*) – Specify how encoding errors should handled.
- ****fmtparams** – The CSV dialect format parameters.

Notes

The dialect and fmtparams expose the Python csv.reader() parameters.

Examples

Get the rows from a CSV file and process them:

```
async for row in axon.csvrows(sha256):
    await dostuff(row)
```

Get the rows from a tab separated file and process them:

```
async for row in axon.csvrows(sha256, delimiter='	'):
    await dostuff(row)
```

Yields
list – Decoded CSV rows.

`async del_(sha256)`

Remove the given bytes from the Axon by sha256.

Parameters
sha256 (*bytes*) – The sha256, in bytes, to remove from the Axon.

Returns
True if the file is removed; false if the file is not present.

Return type
boolean

`async dels(sha256s)`

Given a list of sha256 hashes, delete the files from the Axon.

Parameters
sha256s (*list*) – A list of sha256 hashes in bytes form.

Returns
A list of booleans, indicating if the file was deleted or not.

Return type
list

`async get(sha256, offs=None, size=None)`

Get bytes of a file.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file in bytes.
- **offs** (*int*) – The offset to start reading from.
- **size** (*int*) – The total number of bytes to read.

Examples

Get the bytes from an Axon and process them:

```
buf = b''
async for bytz in axon.get(sha256):
    buf += bytz

await dostuff(buf)
```

Yields
bytes – Chunks of the file bytes.

Raises
synapse.exc.NoSuchFile – If the file does not exist.

async **has**(*sha256*)
Check if the Axon has a file.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
True if the Axon has the file; false otherwise.

Return type
boolean

async **hashes**(*offs*, *wait=False*, *timeout=None*)
Yield hash rows for files that exist in the Axon in added order starting at an offset.

Parameters

- **offs** (*int*) – The index offset.
- **wait** (*boolean*) – Wait for new results and yield them in realtime.
- **timeout** (*int*) – Max time to wait for new results.

Yields
(int, (bytes, int)) – An index offset and the file SHA-256 and size.

async **hashset**(*sha256*)
Calculate additional hashes for a file in the Axon.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
A dictionary containing hashes of the file.

Return type
dict

async **history**(*tick*, *tock=None*)
Yield hash rows for files that existing in the Axon after a given point in time.

Parameters

- **tick** (*int*) – The starting time (in epoch milliseconds).
- **tock** (*int*) – The ending time to stop iterating at (in epoch milliseconds).

Yields
(int, (bytes, int)) – A tuple containing time of the hash was added and the file SHA-256 and size.

`async iterMpkFile(`*sha256*`)`

Yield items from a MsgPack (.mpk) file in the Axon.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Yields
Unpacked items from the bytes.

`async jsonlines(`*sha256*, *errors='ignore'*`)`

Yield JSON objects from JSONL (JSON lines) file.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file.
- **errors** (*str*) – Specify how encoding errors should handled.

Yields
object – Decoded JSON objects.

`async metrics()`

Get the runtime metrics of the Axon.

Returns
A dictionary of runtime data about the Axon.

Return type
dict

`async postfiles(`*fields*, *url*, *params=None*, *headers=None*, *method='POST'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*`)`

`async put(`*byts*`)`

Store bytes in the Axon.

Parameters
byts (*bytes*) – The bytes to store in the Axon.

Notes

This API should not be used for files greater than 128 MiB in size.

Returns
A tuple with the file size and sha256 hash of the bytes.

Return type
tuple(int, bytes)

`async puts(`*files*`)`

Store a set of bytes in the Axon.

Parameters
files (*list*) – A list of bytes to store in the Axon.

Notes

This API should not be used for storing more than 128 MiB of bytes at once.

Returns
A list containing tuples of file size and sha256 hash of the saved bytes.

Return type
list(tuple(int, bytes))

`async readlines(sha256, errors='ignore')`

Yield lines from a multi-line text file in the axon.

Parameters

- **sha256** (*bytes*) – The sha256 hash of the file.
- **errors** (*str*) – Specify how encoding errors should handled.

Yields
str – Lines of text

`async size(sha256)`

Get the size of a file in the Axon.

Parameters
sha256 (*bytes*) – The sha256 hash of the file in bytes.

Returns
The size of the file, in bytes. If not present, None is returned.

Return type
int

`async upload()`

Get an Upload object.

Notes

The UpLoad object should be used to manage uploads greater than 128 MiB in size.

Examples

Use an UpLoad object to upload a file to the Axon:

```
async with axonProxy.upload() as upfd:
    # Assumes bytesGenerator yields bytes
    async for byts in bytsgenerator():
        upfd.write(byts)
    upfd.save()
```

Use a single UpLoad object to save multiple files:

```
async with axonProxy.upload() as upfd:
    for fp in file_paths:
        # Assumes bytesGenerator yields bytes
        async for byts in bytsgenerator(fp):
```

(continues on next page)

(continued from previous page)

```
        upfd.write(byts)
    upfd.save()
```

Returns
An Upload manager object.

Return type
UpLoadShare

async wants(*sha256s*)

Get a list of sha256 values the axon does not have from a input list.

Parameters
sha256s (*list*) – A list of sha256 values as bytes.

Returns
A list of bytes containing the sha256 hashes the Axon does not have.

Return type
list

async wget(*url*, *params=None*, *headers=None*, *json=None*, *body=None*, *method='GET'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

Stream a file download directly into the Axon.

Parameters

- **url** (*str*) – The URL to retrieve.
- **params** (*dict*) – Additional parameters to add to the URL.
- **headers** (*dict*) – Additional HTTP headers to add in the request.
- **json** – A JSON body which is included with the request.
- **body** – The body to be included in the request.
- **method** (*str*) – The HTTP method to use.
- **ssl** (*bool*) – Perform SSL verification.
- **timeout** (*int*) – The timeout of the request, in seconds.
- **ssl_opts** (*dict*) – Additional SSL/TLS options.

Notes

The response body will be stored, regardless of the response code. The ok value in the response does not reflect that a status code, such as a 404, was encountered when retrieving the URL.

The ssl_opts dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl␣
↪argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can␣
↪be included in client_cert.
}
```

The dictionary returned by this may contain the following values:

```
{
    'ok': <boolean> - False if there were exceptions retrieving the URL.
    'url': <str> - The URL retrieved (which could have been redirected). This␣
↪is a url-decoded string.
    'code': <int> - The response code.
    'reason': <str> - The reason phrase for the HTTP status code.
    'mesg': <str> - An error message if there was an exception when retrieving␣
↪the URL.
    'err': <tuple> - An error tuple if there was an exception when retrieving␣
↪the URL.
    'headers': <dict> - The response headers as a dictionary.
    'size': <int> - The size in bytes of the response body.
    'hashes': {
        'md5': <str> - The MD5 hash of the response body.
        'sha1': <str> - The SHA1 hash of the response body.
        'sha256': <str> - The SHA256 hash of the response body.
        'sha512': <str> - The SHA512 hash of the response body.
    },
    'request': {
        'url': The request URL. This is a url-decoded string.
        'headers': The request headers.
        'method': The request method.
    }
    'history': A sequence of response bodies to track any redirects, not␣
↪including hashes.
}
```

Returns
An information dictionary containing the results of the request.

Return type
dict

async wput(*sha256*, *url*, *params=None*, *headers=None*, *method='PUT'*, *ssl=True*, *timeout=None*, *proxy=None*, *ssl_opts=None*)

class synapse.axon.**AxonFileHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonHandlerMixin*, *Handler*

async getAxonInfo()

class synapse.axon.**AxonHandlerMixin**

Bases: object

getAxon()

Get a reference to the Axon interface used by the handler.

class synapse.axon.**AxonHttpBySha256InvalidV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonFileHandler*

async delete(*sha256*)

async get(*sha256*)

async head(*sha256*)

class synapse.axon.**AxonHttpBySha256V1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonFileHandler*

async delete(*sha256*)

async get(*sha256*)

async head(*sha256*)

class synapse.axon.**AxonHttpDelV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonHandlerMixin*, *Handler*

async post()

class synapse.axon.**AxonHttpHasV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonHandlerMixin*, *Handler*

async get(*sha256*)

class synapse.axon.**AxonHttpUploadV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *AxonHandlerMixin*, *StreamHandler*

async data_received(*chunk*)

Implement this method to handle streamed request data.

Requires the *.stream_request_body* decorator.

May be a coroutine for flow control.

on_connection_close()

Called in async handlers if the client closed the connection.

Override this to clean up resources associated with long-lived connections. Note that this method is called only if the connection was closed during asynchronous processing; if you need to do cleanup after every request override *on_finish* instead.

Proxies may keep a connection open for a time (perhaps indefinitely) after the client has gone away, so this method may not be called promptly after the end user closes their connection.

on_finish()

Called after the end of a request.

Override this method to perform cleanup, logging, etc. This method is a counterpart to *prepare*. `on_finish` may not produce any output, as it is called after the response has been sent to the client.

async post()

Called after all data has been read.

async prepare()

Called at the beginning of a request before *get*/*post*/etc.

Override this method to perform common initialization regardless of the request method.

Asynchronous support: Use `async def` or decorate this method with *.gen.coroutine* to make it asynchronous. If this method returns an `Awaitable` execution will not proceed until the `Awaitable` is done.

New in version 3.1: Asynchronous support.

`async put()`

`class synapse.axon.UpLoad`

Bases: *Base*

An object used to manage uploads to the Axon.

`async save()`

Save the currently uploaded bytes to the Axon.

Notes

This resets the Upload object, so it can be reused.

Returns
A tuple of sizes in bytes and the sha256 hash of the saved files.

Return type
tuple(int, bytes)

`async write(byts)`

Write bytes to the Upload object.

Parameters
byts (*bytes*) – Bytes to write to the current Upload object.

Returns
Returns None.

Return type
(None)

`class synapse.axon.UpLoadProxy`

Bases: *Share*

`async save()`

`async write(byts)`

`class synapse.axon.UpLoadShare`

Bases: *UpLoad*, *Share*

`typename = 'upload'`

10.1.4 synapse.cells module

10.1.5 synapse.common module

`class synapse.common.NoValu`

Bases: `object`

`async synapse.common.agen(*items)`

`async synapse.common.aspin(genr)`

Async version of spin

synapse.common.**buid**(*valu=None*)

A binary GUID like sequence of 32 bytes.

Parameters

- **valu** (*object*) – Optional, if provided, the hash of the msgpack
- **to** (*encoded form of the object is returned. This can be used*) –
- **buids.** (*create stable*) –

Notes

By default, this returns a random 32 byte value.

Returns
A 32 byte value.

Return type
bytes

synapse.common.**chunks**(*item*, *size*)

Divide an iterable into chunks.

Parameters

- **item** – Item to slice
- **size** (*int*) – Maximum chunk size.

Notes

This supports Generator objects and objects which support calling the __getitem__() method with a slice object.

Yields
Slices of the item containing up to "size" number of items.

synapse.common.**config**(*conf*, *confdefs*)

Initialize a config dict using the given confdef tuples.

synapse.common.**debase64**(*b*)

synapse.common.**deprdate**(*name*, *date*)

synapse.common.**deprecated**(*name*, *curv='2.x'*, *eolv='3.0.0'*)

synapse.common.**ehex**(*byts*)

Encode a bytes variable to a string using binascii.hexlify.

Parameters
byts (*bytes*) – Bytes to encode.

Returns
A string representing the bytes.

Return type
str

synapse.common.**enbase64**(*b*)

synapse.common.**envbool**(*name*, *defval='false'*)

Resolve an environment variable to a boolean value.

Parameters

- **name** (*str*) – Environment variable to resolve.
- **defval** (*str*) – Default string value to resolve as.

Notes

False values will be consider strings "0" or "false" after lower casing.

Returns
True if the envar is set, false if it is set to a false value.

Return type
boolean

synapse.common.**err**(*e*, *fulltb=False*)

synapse.common.**errinfo**(*name*, *mesg*)

synapse.common.**excinfo**(*e*)

Populate err,errmsg,errtrace info from exc.

synapse.common.**firethread**(*f*)

A decorator for making a function fire a thread.

synapse.common.**flatten**(*item*)

Normalize a primitive object for cryptographic signing.

Parameters
item – The python primitive object to normalize.

Notes

Only None, bool, int, bytes, strings, lists, tuples and dictionaries are acceptable input. List objects will be converted to tuples. Dictionary objects must have keys which can be sorted.

Returns
A new copy of the object.

synapse.common.**gendir**(**paths*, ***opts*)

Return the absolute path of the joining of the arguments, creating a directory at the resulting path if one does not exist.

Performs home directory(~) and environment variable expansion.

Parameters

- ***paths** (*[str,...]*) – A list of path elements
- ****opts** – arguments as kwargs to os.makedirs

synapse.common.**genfile**(**paths*)

Create or open (for read/write) a file path join.

Parameters
***paths** – A list of paths to join together to make the file.

Notes

If the file already exists, the fd returned is opened in `r+b` mode. Otherwise, the fd is opened in `w+b` mode.

The file position is set to the start of the file. The user is responsible for truncating (`fd.truncate()`) if the existing file contents are not desired, or seeking to the end (`fd.seek(0, 2)`) to append.

Returns
A file-object which can be read/written too.

Return type
io.BufferedRandom

`synapse.common.`**`genpath`**`(`*`*paths`*`)`

Return an absolute path of the joining of the arguments as path elements

Performs home directory(~) and environment variable expansion on the joined path

Parameters
`*paths` (*`[str,...]`*) – A list of path elements

Note: All paths used by Synapse operations (i.e. everything but the data) shall use this function or one of its callers before storing as object properties.

`synapse.common.`**`getDirSize`**`(`*`*paths`*`)`

Get the size of a directory.

Parameters
`*paths` (*`str`*) – A list of path elements.

Notes

This is equivalent to `du -B 1 -s` and `du -bs`.

Returns
Tuple of total real and total apparent size of all normal files and directories underneath `*paths` plus `*paths` itself.

Return type
tuple

`synapse.common.`**`getSslCtx`**`(`*`cadir`*`, `*`purpose=Purpose.SERVER_AUTH`*`)`

Create as SSL Context and load certificates from a given directory.

Parameters

- **`cadir`** (*`str`*) – Path to load certificates from.
- **`purpose`** – SSLContext purposes flags.

Returns
A SSL Context object.

Return type
ssl.SSLContext

`synapse.common.`**`getSynDir`**`(`*`*paths`*`)`

`synapse.common.`**`getSynPath`**`(`*`*paths`*`)` synapse.common.**getTempDir**(*dirn=None*)

synapse.common.**getbytes**(**paths*, ***opts*)

synapse.common.**getfile**(**paths*, ***opts*)

Return a file at the path resulting from joining of the arguments, or None if the file does not exist.

Parameters

- ***paths** (*[str,...]*) – A list of path elements
- ****opts** – arguments as kwargs to io.open

Returns
A file-object which can be read/written too.

Return type
io.BufferedRandom

synapse.common.**guid**(*valu=None*)

Get a 16 byte guid value.

By default, this is a random guid value.

Parameters
valu – Object used to construct the guid valu from. This must be able to be msgpack'd.

Returns
32 character, lowercase ascii string.

Return type
str

synapse.common.**httpcodereason**(*code*)

Get the reason for an HTTP status code.

Parameters
code (*int*) – The code.

Note: If the status code is unknown, a string indicating it is unknown is returned.

Returns
A string describing the status code.

Return type
str

synapse.common.**hugeadd**(*x*, *y*)

Add two decimal.Decimal with proper precision to support synapse hugenums.

synapse.common.**hugediv**(*x*, *y*)

Divide two decimal.Decimal with proper precision to support synapse hugenums.

synapse.common.**hugemod**(*x*, *y*)

synapse.common.**hugemul**(*x*, *y*)

Multiply two decimal.Decimal with proper precision to support synapse hugenums.

synapse.common.**hugenum**(*valu*)

Return a decimal.Decimal with proper precision for use as a synapse hugenum.

synapse.common.**hugepow**(*x*, *y*)

Return the first operand to the power of the second operand.

synapse.common.**hugeround**(*x*)

Round a decimal.Decimal with proper precision for synapse hugenums.

synapse.common.**hugescaleb**(*x*, *y*)

Return the first operand with its exponent adjusted by the second operand.

synapse.common.**hugesub**(*x*, *y*)

Subtract two decimal.Decimal with proper precision to support synapse hugenums.

synapse.common.**int64en**(*i*)

Encode an unsigned 64-bit int into 8 byte big-endian bytes

synapse.common.**int64un**(*b*)

Decode an unsigned 64-bit int from 8 byte big-endian

synapse.common.**intify**(*x*)

Ensure (or coerce) a value into being an integer or None.

Parameters

x (*obj*) – An object to intify

Returns

The int value (or None)

Return type

(int)

synapse.common.**isbuidhex**(*text*)

synapse.common.**isguid**(*text*)

synapse.common.**iterfd**(*fd*, *size=10000000*)

Generator which yields bytes from a file descriptor.

Parameters

- **fd** (*file*) – A file-like object to read bytes from.
- **size** (*int*) – Size, in bytes, of the number of bytes to read from the
- **time.** (*fd at a given*) –

Notes

If the first read call on the file descriptor is a empty bytestring, that zero length bytestring will be yielded and the generator will then be exhausted. This behavior is intended to allow the yielding of contents of a zero byte file.

Yields

bytes – Bytes from the file descriptor.

synapse.common.**iterzip**(**args*, *fillvalue=None*)

synapse.common.**jslines**(**paths*)

synapse.common.**jsload**(**paths*)

synapse.common.**jsonsafe_nodeedits**(*nodeedits*)

Hexlify the buid of each node:edits

synapse.common.**jssave**(*js*, **paths*)

synapse.common.**listdir**(**paths*, *glob=None*)

List the (optionally glob filtered) full paths from a dir.

Parameters

- ***paths** (*[str, ...]*) – A list of path elements
- **glob** (*str*) – An optional fnmatch glob str

synapse.common.**makedirs**(*path*, *mode=511*)

async synapse.common.**merggenr**(*genrs*, *cmprkey*)

Iterate multiple sorted async generators and yield their results in order.

Parameters

- **genrs** (*Sequence[AsyncGenerator[T]]*) – a sequence of async generator that each yield sorted items
- **cmprkey** (*Callable[T, T, bool]*) – a comparison function over the items yielded

Note: If the genrs yield increasing items, cmprkey should return True if the first parameter is less than the second parameter, e.g lambda x, y: x < y.

async synapse.common.**merggenr2**(*genrs*, *cmprkey=None*, *reverse=False*)

Optimized version of merggenr based on heapq.merge

synapse.common.**mononow**()

Get the current monotonic clock time in milliseconds.

This relies on time.monotonic_ns(), which is a relative time.

Returns

Monotonic clock time in milliseconds.

Return type

int

synapse.common.**normLogLevel**(*valu*)

Norm a log level value to a integer.

Parameters

valu – The value to norm (a string or integer).

Returns

A valid Logging log level.

Return type

int

synapse.common.**now**()

Get the current epoch time in milliseconds.

This relies on time.time_ns(), which is system-dependent in terms of resolution.

Returns
Epoch time in milliseconds.

Return type
int

synapse.common.**reprauthrule**(*rule*)

synapse.common.**reqJsonSafeStrict**(*item*)

Require the item to be safe to serialize to JSON without type coercion issues.

Parameters
item – The python primitive to check.

Returns
None

Raises
s_exc.BadArg – If the item contains invalid data.

synapse.common.**reqbytes**(**paths*)

synapse.common.**reqdir**(**paths*)

Return the absolute path of the joining of the arguments, raising an exception if a directory does not exist at the resulting path.

Performs home directory(~) and environment variable expansion.

Parameters
***paths** (*[str, ...]*) – A list of path elements

synapse.common.**reqfile**(**paths*, ***opts*)

Return a file at the path resulting from joining of the arguments, raising an exception if the file does not exist.

Parameters

- ***paths** (*[str, ...]*) – A list of path elements
- ****opts** – arguments as kwargs to io.open

Returns
A file-object which can be read/written too.

Return type
io.BufferedRandom

synapse.common.**reqjsonsafe**(*item*)

Returns None if item is json serializable, otherwise raises an exception. Uses default type coercion from built-in json.dumps.

synapse.common.**reqpath**(**paths*)

Return the absolute path of the joining of the arguments, raising an exception if a file doesn't exist at resulting path

Parameters
***paths** (*[str, ...]*) – A list of path elements

synapse.common.**result**(*retn*)

Return a value or raise an exception from a retn tuple.

synapse.common.**retnexc**(*e*)

Construct a retn tuple for the given exception.

synapse.common.**setlogging**(*mlogger*, *defval=None*, *structlog=None*, *log_setup=True*, *datefmt=None*)

Configure synapse logging.

Parameters

- **mlogger** (*logging.Logger*) – Reference to a logging.Logger()
- **defval** (*str*) – Default log level. May be an integer.
- **structlog** (*bool*) – Enabled structured (jsonl) logging output.
- **datefmt** (*str*) – Optional strftime format string.

Notes

This calls logging.basicConfig and should only be called once per process.

Returns
None

synapse.common.**signedint64en**(*i*)

Encode a signed 64-bit int into 8 byte big-endian bytes

synapse.common.**signedint64un**(*b*)

Decode a signed 64-bit int from 8 byte big-endian

synapse.common.**spin**(*genr*)

Crank through a generator but discard the yielded values.

Parameters
genr – Any generator or iterable valu.

Notes

This generator is exhausted via the `collections.dequeue()` constructor with a `maxlen=0`, which will quickly exhaust an iterator staying in C code as much as possible.

Returns
None

synapse.common.**switchext**(**paths*, *ext*)

Return an absolute path of the joining of the arguments with the extension replaced.

If an extension does not exist, it will be added.

Parameters

- ***paths** (*[str,...]*) – A list of path elements
- **ext** (*str*) – A file extension (e.g. '.txt'). It should begin with a period.

synapse.common.**todo**(*_todoname*, **args*, ***kwargs*)

Construct and return a todo tuple of (name, args, kwargs).

Note: the odd name for the first parameter is to avoid collision with keys in kwargs.

synapse.common.**tuplify**(*obj*)

Convert a nested set of python primitives into tupleized forms via msgpack.

synapse.common.**uhex**(*text*)

Decode a hex string into bytes.

Parameters
text (*str*) – Text to decode.

Returns
The decoded bytes.

Return type
bytes

synapse.common.**unjsonsafe_nodeedits**(*nodeedits*)

synapse.common.**verstr**(*vtup*)

Convert a version tuple to a string.

synapse.common.**vertup**(*vstr*)

Convert a version string to a tuple.

Example

ver = vertup('1.3.30')

async synapse.common.**wait_for**(*fut*, *timeout*)

synapse.common.**worker**(*meth*, **args*, ***kwargs*)

synapse.common.**yamlload**(**paths*)

synapse.common.**yamlloads**(*data*)

synapse.common.**yamlmod**(*obj*, **paths*)

Combines/creates a yaml file and combines with obj. obj and file must be maps/dict or empty.

synapse.common.**yamlpop**(*key*, **paths*)

Pop a key out of a yaml file.

Parameters

- **key** (*str*) – Name of the key to remove.
- ***paths** – Path to a yaml file. The file must be a map / dictionary.

Returns
None

synapse.common.**yamlsave**(*obj*, **paths*)

10.1.6 synapse.cortex module

class synapse.cortex.**CoreApi**

Bases: *CellApi*

The CoreApi is exposed when connecting to a Cortex over Telepath.

Many CoreApi methods operate on packed nodes consisting of primitive data structures which can be serialized with msgpack/json.

An example of a packaged Node:

```
( (<form>, <valu>), {

    "props": {
        <name>: <valu>,
        ...
    },
    "tags": {
        "foo": <time>,
        "foo.bar": <time>,
    },
})
```

async **addFeedData**(*name*, *items*, *, *viewiden=None*)

async **addForm**(*formname*, *basetype*, *typeopts*, *typeinfo*)

Add an extended form to the data model.

Extended forms *must* begin with _

async **addFormProp**(*form*, *prop*, *tdef*, *info*)

Add an extended property to the given form.

Extended properties *must* begin with _

async **addNode**(*form*, *valu*, *props=None*)

Deprecated in 2.0.0.

async **addNodes**(*nodes*)

Add a list of packed nodes to the cortex.

Parameters

nodes (*list*) – [((form, valu), {'props':{}, 'tags':{}}), …]

Yields

(tuple) – Packed node tuples ((form,valu), {'props': {}, 'tags':{}})

Deprecated in 2.0.0

addStormDmon(*ddef*)

async **addStormPkg**(*pkgdef*, *verify=False*)

async **addTagProp**(*name*, *tdef*, *info*)

Add a tag property to record data about tags on nodes.

async **addUnivProp**(*name*, *tdef*, *info*)

Add an extended universal property.

Extended properties *must* begin with _

addUserNotif(*useriden*, *mesgtype*, *mesgdata=None*)

bumpStormDmon(*iden*)

async **callStorm**(*text*, *opts=None*)

Return the value expressed in a return() statement within storm.

cloneLayer(*iden*, *ldef=None*)

`async count(text, opts=None)`
Count the number of nodes which result from a storm query.

Parameters

- **text** (*str*) – Storm query text.
- **opts** (*dict*) – Storm query options.

Returns
The number of nodes resulting from the query.

Return type
(int)

`async delForm(formname)`
Remove an extended form from the data model.

`async delFormProp(form, name)`
Remove an extended property from the given form.

`delStormDmon(iden)`

`async delStormPkg(iden)`

`async delTagProp(name)`
Remove a previously added tag property.

`async delUnivProp(name)`
Remove an extended universal property.

`delUserNotif(indx)`

`disableMigrationMode()`

`disableStormDmon(iden)`

`enableMigrationMode()`

`enableStormDmon(iden)`

`async exportStorm(text, opts=None)`
Execute a storm query and package nodes for export/import.

NOTE: This API yields nodes after an initial complete lift
in order to limit exported edges.

`async feedFromAxon(sha256, opts=None)`
Import a msgpack .nodes file from the axon.

`async getAxonBytes(sha256)`

`async getAxonUpload()`

`getCoreInfo()`
Return static generic information about the cortex including model definition

`async getCoreInfoV2()`
Return static generic information about the cortex including model definition

`getCoreMods()`

async **getFeedFuncs**()

Get a list of Cortex feed functions.

Notes

Each feed dictionary has the name of the feed function, the full docstring for the feed function, and the first line of the docstring broken out in their own keys for easy use.

Returns
A tuple of dictionaries.

Return type
tuple

getHttpExtApiByPath(*path*)

async **getModelDefs**()

async **getModelDict**()

Return a dictionary which describes the data model.

Returns
A model description dictionary.

Return type
(dict)

async **getPropNorm**(*prop*, *valu*)

Get the normalized property value based on the Cortex data model.

Parameters

- **prop** (*str*) – The property to normalize.
- **valu** – The value to normalize.

Returns
A two item tuple, containing the normed value and the info dictionary.

Return type
(tuple)

Raises

- **s_exc.NoSuchProp** – If the prop does not exist.
- **s_exc.BadTypeValu** – If the value fails to normalize.

getStormDmon(*iden*)

getStormDmonLog(*iden*)

getStormDmons()

getStormPkg(*name*)

getStormPkgs()

async **getStormVar**(*name*, *default=None*) async **getTypeNorm**(*name*, *valu*)

Get the normalized type value based on the Cortex data model.

Parameters

- **name** (*str*) – The type to normalize.
- **valu** – The value to normalize.

Returns
A two item tuple, containing the normed value and the info dictionary.

Return type
(tuple)

Raises

- **s_exc.NoSuchType** – If the type does not exist.
- **s_exc.BadTypeValu** – If the value fails to normalize.

getUserNotif(*indx*)

async **iterFormRows**(*layriden*, *form*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes of a single form, optionally (re)starting at startvalue

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **form** (*str*) – A form name
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

async **iterPropRows**(*layriden*, *form*, *prop*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes with a particular secondary property, optionally (re)starting at startvalue

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **form** (*str*) – A form name.
- **prop** (*str*) – A secondary property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

async **iterTagPropRows**(*layriden*, *tag*, *prop*, *form=None*, *stortype=None*, *startvalu=None*)

Yields (buid, valu) that match a tag:prop, optionally (re)starting at startvalu.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **tag** (*str*) – tag name

- **prop** (*str*) – prop name
- **form** (*Optional[str]*) – optional form name
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

async **iterTagRows**(*layriden*, *tag*, *form=None*, *starttupl=None*)

Yields (buid, (valu, form)) values that match a tag and optional form, optionally (re)starting at starttupl.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **tag** (*str*) – the tag to match
- **form** (*Optional[str]*) – if present, only yields buids of nodes that match the form.
- **starttupl** (*Optional[Tuple[buid, form]]*) – if present, (re)starts the stream of values there.

Returns
AsyncIterator[Tuple(buid, (valu, form))]

Note: This yields (buid, (tagvalu, form)) instead of just buid, valu in order to allow resuming an interrupted call by feeding the last value retrieved into starttupl

async **iterUnivRows**(*layriden*, *prop*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes with a particular universal property, optionally (re)starting at startvalue

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **prop** (*str*) – A universal property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

iterUserNotifs(*useriden*, *size=None*)

async **popStormVar**(*name*, *default=None*)

async **reqValidStorm**(*text*, *opts=None*)

Parse a Storm query to validate it.

Parameters

- **text** (*str*) – The text of the Storm query to parse.
- **opts** (*dict*) – A Storm options dictionary.

Returns
If the query is valid.

Return type
True

Raises
`BadSyntaxError` – If the query is invalid.

`saveLayerNodeEdits`(*layriden*, *edits*, *meta*)

`async setStormVar`(*name*, *valu*)

`async storm`(*text*, *opts=None*)

Evaluate a storm query and yield result messages.

Yields
((str,dict)) – Storm messages.

`async syncIndexEvents`(*matchdef*, *offsdict=None*, *wait=True*)

`async syncLayerNodeEdits`(*offs*, *layriden=None*, *wait=True*)

Yield (indx, mesg) nodeedit sets for the given layer beginning at offset.

Once caught up, this API will begin yielding nodeedits in real-time. The generator will only terminate on network disconnect or if the consumer falls behind the max window size of 10,000 nodeedit messages.

`async syncLayersEvents`(*offsdict=None*, *wait=True*)

`watchAllUserNotifs`(*offs=None*)

`class synapse.cortex.Cortex`

Bases: *OAuthMixin*, *Cell*

A Cortex implements the synapse hypergraph.

The bulk of the Cortex API lives on the Snap() object which can be obtained by calling Cortex.snap() in a with block. This allows callers to manage transaction boundaries explicitly and dramatically increases performance.

`async addCoreQueue`(*name*, *info*)

`async addCronEdits`(*iden*, *edits*)

Take a dictionary of edits and apply them to the appointment (cron job)

`async addCronJob`(*cdef*)

Add a cron job to the cortex. Convenience wrapper around agenda.add

A cron job is a persistently-stored item that causes storm queries to be run in the future. The specification for the times that the queries run can be one-shot or recurring.

Parameters

- **query** (*str*) – The storm query to execute in the future
- **reqs** (*Union[Dict[str, Union[int, List[int]]], List[Dict[...]]]*) – Either a dict of the fixed time fields or a list of such dicts. The keys are in the set ('year', 'month', 'dayofmonth', 'dayofweek', 'hour', 'minute'. The values must be positive integers, except for the key of 'dayofmonth' in which it may also be a negative integer which represents the number of days from the end of the month with -1 representing the last day of the month. All values may also be lists of valid values.
- **incunit** (*Optional[str]*) – A member of the same set as above, with an additional member 'day'. If is None (default), then the appointment is one-shot and will not recur.
- **incvals** (*Union[int, List[int]*) – A integer or a list of integers of the number of units

Returns (bytes):
An iden that can be used to later modify, query, and delete the job.

Notes

reqs must have fields present or incunit must not be None (or both) The incunit if not None it must be larger in unit size than all the keys in all reqs elements. Non-recurring jobs may also have a req of 'now' which will cause the job to also execute immediately.

async **addExtModel**(*model*)

Add an extended model definition to a Cortex from the output of getExtModel().

Parameters
model (*dict*) – An extended model dictionary.

Returns
True when the model was added.

Return type
Bool

Raises

- **s_exc.BadFormDef** – If a form exists with a different definition the provided definition.
- **s_exc.BadPropDef** – If a propery, tagprop, or universal propert from exists with a different definition than the provided definition.

async **addFeedData**(*name*, *items*, *, *viewiden=None*)

Add data using a feed/parser function.

Parameters

- **name** (*str*) – The name of the feed record format.
- **items** (*list*) – A list of items to ingest.
- **viewiden** (*str*) – The iden of a view to use. If a view is not specified, the default view is used.

async **addForm**(*formname*, *basetype*, *typeopts*, *typeinfo*)

async **addFormProp**(*form*, *prop*, *tdef*, *info*)

async **addHttpExtApi**(*adef*)

async **addLayer**(*ldef=None*, *nexs=True*)

Add a Layer to the cortex.

Parameters

- **ldef** (*Optional[Dict]*) – layer configuration
- **nexs** (*bool*) – whether to record a nexus transaction (internal use only)

async **addLayrPull**(*layriden*, *pdef*)

async **addLayrPush**(*layriden*, *pdef*)

async **addNode**(*user*, *form*, *valu*, *props=None*) async addNodeTag(*user*, *iden*, *tag*, *valu=(None, None)*)

Add a tag to a node specified by iden.

Parameters

- **iden** (*str*) – A hex encoded node BUID.
- **tag** (*str*) – A tag string.
- **valu** (*tuple*) – A time interval tuple or (None, None).

async addNodes(*nodedefs*, *view=None*)

Quickly add/modify a list of nodes from node definition tuples. This API is the simplest/fastest way to add nodes, set node props, and add tags to nodes remotely.

Parameters
nodedefs (*list*) – A list of node definition tuples. See below.

A node definition tuple is defined as:

((form, valu), {'props':{}, 'tags':{})

The "props" or "tags" keys may be omitted.

addRuntLift(*prop*, *func*)

Register a runt lift helper for a given prop.

Parameters

- **prop** (*str*) – Full property name for the prop to register the helper for.
- **func** –

Returns
None.

Return type
None

addRuntPropDel(*full*, *func*)

Register a prop set helper for a runt form

addRuntPropSet(*full*, *func*)

Register a prop set helper for a runt form

addStormCmd(*ctor*)

Add a synapse.lib.storm.Cmd class to the cortex.

async addStormDmon(*ddef*)

Add a storm dmon task.

async addStormGraph(*gdef*, *user=None*)

addStormLib(*path*, *ctor*)

async addStormMacro(*mdef*, *user=None*)

async addStormPkg(*pkgdef*, *verify=False*)

Add the given storm package to the cortex.

This will store the package for future use.

async **addStormSvc**(*sdef*)

Add a registered storm service to the cortex.

async **addTagProp**(*name*, *tdef*, *info*)

async **addUnivProp**(*name*, *tdef*, *info*)

async **addUserNotif**(*useriden*, *mesgtype*, *mesgdata=None*)

async **addVault**(*vdef*)

Create a new vault.

Parameters

vdef (*dict*) – The vault to add.

Raises

- *synapse.exc.SchemaViolation* – *vdef* does not conform to the vault schema.
- *synapse.exc.DupName* –
  - Vault already exists for type/scope/owner. - Vault already exists with specified name.
- *synapse.exc.BadArg* –
  - Invalid vault definition provided. - Owner required for unscoped, user, and role vaults. - Vault secrets is not msgpack safe. - Vault configs is not msgpack safe.

Returns: iden of new vault

async **addView**(*vdef*, *nexs=True*)

async **bumpStormDmon**(*iden*)

async **callStorm**(*text*, *opts=None*)

cellapi

alias of *CoreApi*

async **cloneLayer**(*iden*, *ldef=None*)

Make a copy of a Layer in the cortex.

Parameters

- **iden** (*str*) – Layer iden to clone
- **ldef** (*Optional[Dict]*) – Layer configuration overrides

Note: This should only be called with a reasonably static Cortex due to possible races.

confbase = {'_log_conf': {'description': 'Opaque structure used for logging by spawned processes.', 'hideconf': True, 'type': 'object'}, 'aha:admin': {'description': 'An AHA client certificate CN to register as a local admin user.', 'type': 'string'}, 'aha:leader': {'description': 'The AHA service name to claim as the active instance of a storm service.', 'type': 'string'}, 'aha:name': {'description': 'The name of the cell service in the aha service registry.', 'type': 'string'}, 'aha:network': {'description': 'The AHA service network. This makes aha:name/aha:leader relative names.', 'type': 'string'}, 'aha:provision': {'description': 'The telepath URL of the aha provisioning service.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:registry': {'description': 'The telepath URL of the aha service registry.', 'items': {'type': 'string'}, 'type': ['string', 'array']}, 'aha:svcinfo': {'description': 'An AHA svcinfo object. If set, this overrides self discovered Aha service information.', 'hidecmdl': True, 'hidedocs': True, 'properties': {'urlinfo': {'properties': {'host': {'type': 'string'}, 'port': {'type': 'integer'}, 'schema': {'type': 'string'}}, 'required': ('host', 'port', 'scheme'), 'type': 'object'}}, 'required': ('urlinfo',), 'type': 'object'}, 'aha:user': {'description': 'The username of this service when connecting to others.', 'type': 'string'}, 'auth:anon': {'description': 'Allow anonymous telepath access by mapping to the given user name.', 'type': 'string'}, 'auth:conf': {'description': 'Extended configuration to be used by an alternate auth constructor.', 'hideconf': True, 'type': 'object'}, 'auth:ctor': {'description': 'Allow the construction of the cell auth object to be hooked at runtime.', 'hideconf': True, 'type': 'string'}, 'auth:passwd': {'description': 'Set to <passwd> (local only) to bootstrap the root user password.', 'type': 'string'}, 'backup:dir': {'description': 'A directory outside the service directory where backups will be saved. Defaults to ./backups in the service storage directory.', 'type': 'string'}, 'cell:ctor': {'description': 'An optional python path to the Cell class. Used by stemcell.', 'hideconf': True, 'type': 'string'}, 'cell:guid': {'description': 'An optional hard-coded GUID to store as the permanent GUID for the service.', 'hideconf': True, 'type': 'string'}, 'dmon:listen': {'description': 'A config-driven way to specify the telepath bind URL.', 'type': ['string', 'null']}, 'https:headers': {'description': 'Headers to add to all HTTPS server responses.', 'hidecmdl': True, 'type': 'object'}, 'https:parse:proxy:remoteip': {'default': False, 'description': 'Enable the HTTPS server to parse X-Forwarded-For and X-Real-IP headers to determine requester IP addresses.', 'type': 'boolean'}, 'https:port': {'description': 'A config-driven way to specify the HTTPS port.', 'type': ['integer', 'null']}, 'inaugural': {'description': 'Data used to drive configuration of the service upon first startup.', 'hidedocs': True, 'properties': {'roles': {'items': {'additionalProperties': False, 'properties': {'name': {'pattern': '^(?!all$).+$', 'type': 'string'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}, 'users': {'items': {'additionalProperties': False, 'properties': {'admin': {'default': False, 'type': 'boolean'}, 'email': {'type': 'string'}, 'name': {'pattern': '^(?!root$).+$', 'type': 'string'}, 'roles': {'items': {'type': 'string'}, 'type': 'array'}, 'rules': {'items': {'items': [{'type': 'boolean'}, {'type': 'array', 'items': {'type': 'string'}}], 'maxItems': 2, 'minItems': 2, 'type': 'array'}, 'type': 'array'}}, 'required': ['name'], 'type': 'object'}, 'type': 'array'}}, 'type': 'object'}, 'limit:disk:free': {'default': 5, 'description': 'Minimum disk free space percentage before setting the cell read-only.', 'maximum': 100, 'minimum': 0, 'type': ['integer', 'null']}, 'max:users': {'default': 0, 'description': 'Maximum number of users allowed on system, not including root or locked/archived users (0 is no limit).', 'minimum': 0, 'type': 'integer'}, 'mirror': {'description': 'A telepath URL for our upstream mirror (we must be a backup!).', 'hidecmdl': False, 'hidedocs': False, 'type': ['string', 'null']}, 'nexslog:async': {'default': True, 'description': 'Set to false to disable async memory mapping of the nexus change log.', 'hidecmdl': True, 'hidedocs': True, 'type': 'boolean'}, 'nexslog:en': {'default': True, 'description': 'Record all

confdefs = {'axon': {'description': 'A telepath URL for a remote axon.', 'type': 'string'}, 'cron:enable': {'default': True, 'description': 'Deprecated. This option no longer controls cron execution and will be removed in Synapse 3.0.', 'type': 'boolean'}, 'http:proxy': {'description': 'An aiohttp-socks compatible proxy URL to use storm HTTP API.', 'type': 'string'}, 'jsonstor': {'description': 'A telepath URL for a remote jsonstor.', 'type': 'string'}, 'layer:lmdb:map_async': {'default': True, 'description': 'Set the default lmdb:map_async value in LMDB layers.', 'type': 'boolean'}, 'layer:lmdb:max_replay_log': {'default': 10000, 'description': 'Set the max size of the replay log for all layers.', 'type': 'integer'}, 'layers:lockmemory': {'default': False, 'description': 'Should new layers lock memory for performance by default.', 'type': 'boolean'}, 'layers:logedits': {'default': True, 'description': 'Whether nodeedits are logged in each layer.', 'type': 'boolean'}, 'max:nodes': {'description': 'Maximum number of nodes which are allowed to be stored in a Cortex.', 'hidecmdl': True, 'minimum': 1, 'type': 'integer'}, 'modules': {'default': [], 'description': 'A list of module classes to load.', 'type': 'array'}, 'provenance:en': {'default': False, 'description': 'This no longer does anything.', 'hideconf': True, 'type': 'boolean'}, 'storm:interface:scrape': {'default': True, 'description': 'Enable Storm scrape interfaces when using $lib.scrape APIs.', 'type': 'boolean'}, 'storm:interface:search': {'default': True, 'description': 'Enable Storm search interfaces for lookup mode.', 'type': 'boolean'}, 'storm:log': {'default': False, 'description': 'Log storm queries via system logger.', 'type': 'boolean'}, 'storm:log:level': {'default': 'INFO', 'description': 'Logging log level to emit storm logs at.', 'type': ['integer', 'string']}, 'tls:ca:dir': {'description': 'An optional directory of CAs which are added to the TLS CA chain for Storm HTTP API calls.', 'type': 'string'}, 'trigger:enable': {'default': True, 'description': 'Deprecated. This option no longer controls trigger execution and will be removed in Synapse 3.0.', 'type': 'boolean'}}

async **coreQueueCull**(*name*, *offs*)

async **coreQueueGet**(*name*, *offs=0*, *cull=True*, *wait=False*)

async **coreQueueGets**(*name*, *offs=0*, *cull=True*, *wait=False*, *size=None*)

async **coreQueuePop**(*name*, *offs*)

async **coreQueuePuts**(*name*, *items*)

async **coreQueueSize**(*name*)

async **count**(*text*, *opts=None*)

async **delCoreQueue**(*name*)

async **delCronJob**(*iden*)

Delete a cron job

Parameters

iden (*bytes*) – The iden of the cron job to be deleted

async **delForm**(*formname*)

async **delFormProp**(*form*, *prop*)

async **delHttpExtApi**(*iden*)

async **delJsonObj**(*path*)

async **delJsonObjProp**(*path*, *prop*)

async **delLayer**(*iden*)

async **delLayrPull**(*layriden*, *pulliden*)

async **delLayrPush**(*layriden*, *pushiden*)

async **delNodeTag**(*user*, *iden*, *tag*)

Delete a tag from the node specified by iden.

Parameters

- **iden** (*str*) – A hex encoded node BUID.
- **tag** (*str*) – A tag string.

async **delStormCmd**(*name*)

Remove a previously set pure storm command.

async **delStormDmon**(*iden*)

Stop and remove a storm dmon.

async **delStormGraph**(*iden*, *user=None*)

async **delStormMacro**(*name*, *user=None*)

async **delStormPkg**(*name*)

async **delStormPool**()

async **delStormSvc**(*iden*)

async **delTagModel**(*tagname*)

Delete all the model specification properties for a tag.

Parameters

tagname (*str*) – The name of the tag.

async **delTagProp**(*name*)

async **delUnivProp**(*prop*)

async **delUserNotif**(*indx*)

async **delVault**(*iden*)

Delete a vault.

Parameters

iden (*str*) – Iden of the vault to delete.

Returns: None

async **delView**(*iden*)

`async disableCronJob(iden)`

Enable a cron job

Parameters
`iden` (*bytes*) – The iden of the cron job to be changed

`async disableStormDmon(iden)`

`async editCronJob(iden, name, valu)`

Modify a cron job definition.

`async enableCronJob(iden)`

Enable a cron job

Parameters
`iden` (*bytes*) – The iden of the cron job to be changed

`async enableStormDmon(iden)`

`enterMigrationMode()`

`async exportStorm(text, opts=None)`

`async exportStormToAxon(text, opts=None)`

`async feedFromAxon(sha256, opts=None)`

`async finiStormPool()`

`async getAxon()`

`async getCellApi(link, user, path)`

Get an instance of the telepath Client object for a given user, link and path.

Parameters

- `link` (*s_link.Link*) – The link object.
- `user` (*s_hive.HiveUser*) – The heavy user object.
- `path` (*str*) – The path requested.

Notes

This defaults to the self.cellapi class. Implementors may override the default class attribute for cellapi to share a different interface.

Returns
The shared object for this cell.

Return type
object

`getCoreInfo()`

This API is deprecated.

`async getCoreInfoV2()`

`getCoreMod(name)`

`getCoreMods()`

`async getCoreQueue(`*name*`)`

`getDataModel()`

`async getDeprLocks()`

Return a dictionary of deprecated properties and their lock status.

`async getExtModel()`

Get all extended model properties in the Cortex.

Returns
A dictionary containing forms, form properties, universal properties and tag properties.

Return type
dict

`getFeedFunc(`*name*`)`

Get a data ingest function.

`async getFeedFuncs()`

`async getFormCounts()`

Return total form counts for all existing layers

`async getHttpExtApi(`*iden*`)`

`async getHttpExtApiByPath(`*path*`)`

`async getHttpExtApis()`

`async getJsonObj(`*path*`)`

`async getJsonObjProp(`*path*, *prop*`)`

`async getJsonObjs(`*path*`)`

`getLayer(`*iden=None*`)`

Get a Layer object.

Parameters
iden (*str*) – The layer iden to retrieve.

Returns
A Layer object.

Return type
Layer

`async getLayerDef(`*iden=None*`)`

`async getLayerDefs()`

`async getModelDefs()`

`async getModelDict()`

`async getNodeByNdef(`*ndef*, *view=None*`)`

Return a single Node() instance by (form,valu) tuple.

async getPropNorm(*prop*, *valu*)

Get the normalized property value based on the Cortex data model.

Parameters

- prop (*str*) – The property to normalize.
- valu – The value to normalize.

Returns

A two item tuple, containing the normed value and the info dictionary.

Return type

(tuple)

Raises

- s_exc.NoSuchProp – If the prop does not exist.
- s_exc.BadTypeValu – If the value fails to normalize.

getStormCmd(*name*)

getStormCmds()

async getStormDmon(*iden*)

async getStormDmonLog(*iden*)

async getStormDmons()

async getStormDocs()

Get a struct containing the Storm Types documentation.

Returns

A Dictionary of storm documentation information.

Return type

dict

async getStormGraph(*iden*, *user=None*)

async getStormGraphs(*user=None*)

async getStormIfaces(*name*)

getStormLib(*path*)

getStormMacro(*name*, *user=None*)

async getStormMacros(*user=None*)

async getStormMod(*name*, *reqvers=None*)

async getStormMods()

async getStormPkg(*name*)

async getStormPkgs()

async getStormPool()

async getStormQuery(*text*, *mode='storm'*) `getStormRuntime`(*query*, *opts=None*)

`getStormSvc`(*name*)

`getStormSvcs`()

`async getStormVar`(*name*, *default=None*)

`async getTagModel`(*tagname*)

Retrieve the tag model specification for a tag.

Returns
The tag model specification or None.

Return type
(dict)

`async getTagPrune`(*tagname*)

`async getTypeNorm`(*name*, *valu*)

Get the normalized type value based on the Cortex data model.

Parameters

- **name** (*str*) – The type to normalize.
- **valu** – The value to normalize.

Returns
A two item tuple, containing the normed value and the info dictionary.

Return type
(tuple)

Raises

- **s_exc.NoSuchType** – If the type does not exist.
- **s_exc.BadTypeValu** – If the value fails to normalize.

`async getUserNotif`(*indx*)

`getVault`(*iden*)

Get a vault.

Parameters
iden (*str*) – Iden of the vault to get.

Returns: vault or None

`getVaultByName`(*name*)

Get a vault by name.

Parameters
name (*str*) – Name of the vault to get.

Returns: vault or None

`getVaultByType`(*vtype*, *useriden*, *scope=None*)

Get a vault of type *vtype* and scope *scope* for user with *iden*.

This function allows the caller to retrieve a vault of the specified *vtype* by searching for the first available vault that matches the *vtype* and *scope* criteria. The search order for opening vaults is as follows:

- If *scope* is specified, return the vault with *vtype* and *scope*. Return None if such a vault doesn't exist.
- Check 'user' scope for a vault of *vtype*. Continue if non-existent.
- Check 'role' scope for a vault of *vtype*. Continue if non-existent.
- Check 'global' scope for a vault of *vtype*. Continue if non-existent.
- Return None

Parameters

- **vtype** (*str*) – Type of the vault to open.
- **useriden** (*str*) – Iden of user trying to open the vault.
- **scope** (*str | None*) – The vault scope to open.

Raises

synapse.exc.BadArg – Invalid scope specified.

Returns: vault or None if matching vault could not be found.

`getView(iden=None, user=None)`

Get a View object.

Parameters

iden (*str*) – The View iden to retrieve.

Returns

A View object.

Return type

View

`async getViewDef(iden)`

`async getViewDefs(deporder=False)`

`async hasJsonObj(path)`

`hiveapi`

alias of `HiveApi`

`async initServiceActive()`

`async initServicePassive()`

`async initServiceRuntime()`

`async initServiceStorage()`

`async initStormPool()`

`isTagValid(tagname)`

Check if a tag name is valid according to tag model regular expressions.

Returns

True if the tag is valid.

Return type

(bool)

async itemsStormVar()

async iterFormRows(*layriden*, *form*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes of a single form, optionally (re)starting at startvalu.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **form** (*str*) – A form name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns

AsyncIterator[Tuple(buid, valu)]

async iterPropRows(*layriden*, *form*, *prop*, *stortype=None*, *startvalu=None*)

Yields buid, valu tuples of nodes with a particular secondary property, optionally (re)starting at startvalu.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **form** (*str*) – A form name.
- **prop** (*str*) – A universal property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns

AsyncIterator[Tuple(buid, valu)]

async iterTagPropRows(*layriden*, *tag*, *prop*, *form=None*, *stortype=None*, *startvalu=None*)

Yields (buid, valu) that match a tag:prop, optionally (re)starting at startvalu.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **tag** (*str*) – tag name
- **prop** (*str*) – prop name
- **form** (*Optional[str]*) – optional form name
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns

AsyncIterator[Tuple(buid, valu)]

async iterTagRows(*layriden*, *tag*, *form=None*, *starttupl=None*)

Yields (buid, (valu, form)) values that match a tag and optional form, optionally (re)starting at starttupl.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **tag** (*str*) – the tag to match

- **form** (*Optional[str]*) – if present, only yields buids of nodes that match the form.
- **starttupl** (*Optional[Tuple[buid, form]]*) – if present, (re)starts the stream of values there.

Returns
AsyncIterator[Tuple(buid, (valu, form))]

Note: This yields (buid, (tagvalu, form)) instead of just buid, valu in order to allow resuming an interrupted call by feeding the last value retrieved into starttupl

`async iterUnivRows(layriden, prop, stortype=None, startvalu=None)`

Yields buid, valu tuples of nodes with a particular universal property, optionally (re)starting at startvalu.

Parameters

- **layriden** (*str*) – Iden of the layer to retrieve the nodes
- **prop** (*str*) – A universal property name.
- **stortype** (*Optional[int]*) – a STOR_TYPE_* integer representing the type of form:prop
- **startvalu** (*Any*) – The value to start at. May only be not None if stortype is not None.

Returns
AsyncIterator[Tuple(buid, valu)]

`async iterUserNotifs(useriden, size=None)`

`layerapi`

alias of *LayerApi*

`async classmethod layrctor(*args, **kwargs)`

`async listCoreQueues()`

`async listCronJobs()`

Get information about all the cron jobs accessible to the current user

`listLayers()`

`async listTagModel()`

Retrieve a list of the tag model specifications.

Returns
A list of tag model specification tuples.

Return type
([(str, dict), ...])

`listVaults()`

List all vaults.

Args: None

Raises: None

Yields: tuples of vault info: (<iden>, <name>, <type>, <scope>).

`listViews()`

async **loadCoreModule**(*ctor*, *conf=None*)

Load a single cortex module with the given ctor and conf.

Parameters

- **ctor** (*str*) – The python module class path
- **conf** (*dict*) – Config dictionary for the module

async **loadStormPkg**(*pkgdef*)

Load a storm package into the storm library for this cortex.

NOTE: This will *not* persist the package (allowing service dynamism).

async **modHttpExtApi**(*iden*, *name*, *valu*)

async **modStormGraph**(*iden*, *info*, *user=None*)

async **modStormMacro**(*name*, *info*, *user=None*)

async **moveCronJob**(*useriden*, *croniden*, *viewiden*)

async **nodes**(*text*, *opts=None*)

A simple non-streaming way to return a list of nodes.

async **popStormVar**(*name*, *default=None*)

async **popTagModel**(*tagname*, *name*)

Pop a property from the model specification of a tag.

Parameters

- **tagname** (*str*) – The name of the tag.
- **name** (*str*) – The name of the specification property.

Returns

The current value of the property.

Return type

(object)

async **renameVault**(*iden*, *name*)

Rename a vault.

Parameters

- **iden** (*str*) – Iden of the vault to rename.
- **name** (*str*) – New vault name.

Raises

- *synapse.exc.NoSuchIden* – Vault with *iden* does not exist.
- *synapse.exc.DupName* – Vault with *name* already exists.

Returns: Updated vault.

async **replaceVaultConfigs**(*iden*, *valu*)

Replace the entire vault config.

Parameters

- **iden** (*str*) – The iden of the vault to edit.

- **valu** (*str*) – New configs object to store on the vault.

Raises

- *synapse.exc.BadArg* – *valu* is not a dictionary.
- *synapse.exc.NoSuchIden* – Vault with *iden* does not exist.
- *synapse.exc.NotMsgpackSafe* – *valu* is not msgpack safe.

Returns: New configs.

async **replaceVaultSecrets**(*iden*, *valu*)

Replace the entire vault config.

Parameters

- **iden** (*str*) – The iden of the vault to edit.
- **valu** (*str*) – New secrets object to store on the vault.

Raises

- *synapse.exc.BadArg* – *valu* is not a dictionary.
- *synapse.exc.NoSuchIden* – Vault with *iden* does not exist.
- *synapse.exc.NotMsgpackSafe* – *valu* is not msgpack safe.

Returns: New secrets.

reqStormMacro(*name*, *user=None*)

async **reqValidStorm**(*text*, *opts=None*)

Parse a storm query to validate it.

Parameters

- **text** (*str*) – The text of the Storm query to parse.
- **opts** (*dict*) – A Storm options dictionary.

Returns
If the query is valid.

Return type
True

Raises
BadSyntaxError – If the query is invalid.

async **reqValidStormGraph**(*gdef*)

reqVault(*iden*)

Get a vault by iden.

Parameters
iden (*str*) – Iden of the vault to get.

Raises
synapse.exc.NoSuchIden – Vault with *iden* not found.

Returns: vault

reqVaultByName(*name*)

Get a vault by name.

Parameters

name (*str*) – Name of the vault to get.

Raises

synapse.exc.NoSuchName – Vault with *name* not found.

Returns: vault

reqVaultByType(*vtype*, *iden*, *scope=None*)

Get a vault by type.

Parameters

- **vtype** (*str*) – Type of the vault to get.
- **iden** (*str*) – Iden of the user or role for the vault type.
- **scope** (*str | None*) – Scope of the vault to get.

Raises

synapse.exc.NoSuchName – Vault with *vtype/iden/scope* not found.

Returns: vault

reqView(*iden*, *mesg=None*)

async **runLayrPull**(*layr*, *pdef*)

async **runLayrPush**(*layr*, *pdef*)

async **runRuntLift**(*full*, *valu=None*, *cmpr=None*, *view=None*)

Execute a runt lift function.

Parameters

- **full** (*str*) – Property to lift by.
- **valu** –
- **cmpr** –

Returns

Yields bytes, list tuples where the list contains a series of
key/value pairs which are used to construct a Node object.

Return type

bytes, list

async **runRuntPropDel**(*node*, *prop*)

async **runRuntPropSet**(*node*, *prop*, *valu*)

async **runStormDmon**(*iden*, *ddef*)

async **runStormSvcEvent**(*iden*, *name*)

async **saveLayerNodeEdits**(*layriden*, *edits*, *meta*)

async **setDeprLock**(*name*, *locked*)

`setFeedFunc`(*name*, *func*)

Set a data ingest function.

def func(snap, items):
loaditems…

async `setHttpApiIndx`(*iden*, *indx*)

async `setJsonObj`(*path*, *item*)

async `setJsonObjProp`(*path*, *prop*, *item*)

async `setStormCmd`(*cdef*)

async `setStormGraphPerm`(*gden*, *scope*, *iden*, *level*, *user=None*)

async `setStormMacroPerm`(*name*, *scope*, *iden*, *level*, *user=None*)

async `setStormPool`(*url*, *opts*)

async `setStormSvcEvents`(*iden*, *edef*)

Set the event callbacks for a storm service. Extends the sdef dict.

Parameters

- **iden** (*str*) – The service iden.
- **edef** (*dict*) – The events definition.

Notes

The edef is formatted like the following:

```
{
    <name> : {
        'storm': <storm>
    }
}
```

where `name` is one of the following items:

add

Run the given storm '*before* the service is first added (a la service.add), but not on a reconnect.

del

Run the given storm *after* the service is removed (a la service.del), but not on a disconnect.

Returns
An updated storm service definition dictionary.

Return type
dict

async `setStormVar`(*name*, *valu*)

async **setTagModel**(*tagname*, *name*, *valu*)

Set a model specification property for a tag.

Parameters

- **tagname** (*str*) – The name of the tag.
- **name** (*str*) – The name of the property.
- **valu** (*object*) – The value of the property.

Tag Model Properties:

regex - A list of None or regular expression strings to match each tag level. prune - A number that determines how many levels of pruning are desired.

Examples

await core.setTagModel("cno.cve", "regex", (None, None, "[0-9]{4}", "[0-9]{5}"))

async **setUserLocked**(*iden*, *locked*)

async **setVaultConfigs**(*iden*, *key*, *valu*)

Set vault config item.

This function sets the *key*:*valu* into the vault configs.

Parameters

- **iden** (*str*) – The iden of the vault to edit.
- **key** (*str*) – Vault secret key.
- **valu** (*str*) – Vault secret value. s_common.novalu to delete a key.

Raises

- *synapse.exc.NoSuchIden* – Vault with *iden* does not exist.
- *synapse.exc.NotMsgpackSafe* – One of *key* or *valu* is not msgpack safe.

Returns: Updated vault.

async **setVaultPerm**(*viden*, *iden*, *level*)

Set vault permissions. :param viden: The iden of the vault to edit. :type viden: str :param iden: Iden of the user/role to add permissions for. :type iden: str :param level: Easy perms level. :type level: int

Raises

synapse.exc.NoSuchIden – Vault with *iden* does not exist.

Returns: Updated vault.

async **setVaultSecrets**(*iden*, *key*, *valu*)

Set vault secret item.

This function sets the *key*:*valu* into the vault secrets.

Parameters

- **iden** (*str*) – The iden of the vault to edit.
- **key** (*str*) – Vault secret key.
- **valu** (*str*) – Vault secret value. s_common.novalu to delete a key.

Raises

- `synapse.exc.NoSuchIden` – Vault with *iden* does not exist.
- `synapse.exc.NotMsgpackSafe` – One of *key* or *valu* is not msgpack safe.

Returns: Updated vault.

async **setViewLayers**(*layers*, *iden=None*)

Parameters

- **layers** (*[str]*) – A top-down list of of layer guids
- **iden** (*str*) – The view iden (defaults to default view).

async **snap**(*user=None*, *view=None*)

Return a transaction object for the default view.

Parameters

- **user** (*str*) – The user to get the snap for.
- **view** (View) – View object to use when making the snap.

Notes

This must be used as an asynchronous context manager.

Returns
A Snap object for the view.

Return type
s_snap.Snap

async **storm**(*text*, *opts=None*)

async **stormlist**(*text*, *opts=None*)

async **syncIndexEvents**(*matchdef*, *offsdict=None*, *wait=True*)

Yield (offs, layriden, <STYPE>, <item>) tuples from the nodeedit logs of all layers starting from the given nexus/layer offset (they are synchronized). Only edits that match the filter in matchdef will be yielded, plus EDIT_PROGRESS (see layer.syncIndexEvents) messages.

The format of the 4th element of the tuple depends on STYPE. STYPE is one of the following constants:

> SYNC_LAYR_ADD: item is an empty tuple () SYNC_LAYR_DEL: item is an empty tuple () SYNC_NODEEDIT: item is (buid, form, ETYPE, VALS, META)) or (None, None, s_layer.EDIT_PROGRESS, (), ())

For edits in the past, events are yielded in offset order across all layers. For current data (wait=True), events across different layers may be emitted slightly out of offset order.

Note: Will not yield any values from layers created with logedits disabled

---

Parameters

- **matchdef** (Dict*[str, Sequence[str]]*) – a dict describing which events are yielded. See layer.syncIndexEvents for matchdef specification.
- **offsdict** (*Optional(*Dict*[str,int])*) – starting nexus/editlog offset by layer iden. Defaults to 0 for unspecified layers or if offsdict is None.

- **wait** (*bool*) – whether to pend and stream value until this layer is fini'd

async **syncLayerNodeEdits**(*iden*, *offs*, *wait=True*)

Yield (offs, mesg) tuples for nodeedits in a layer.

async **syncLayersEvents**(*offsdict=None*, *wait=True*)

Yield (offs, layriden, STYP, item, meta) tuples for nodeedits for *all* layers, interspersed with add/del layer messages.

STYP is one of the following constants:
SYNC_NODEEDITS: item is a nodeedits (buid, form, edits) SYNC_LAYR_ADD: A layer was added (item and meta are empty) SYNC_LAYR_DEL: A layer was deleted (item and meta are empty)

Parameters

- **offsdict** (*Optional(Dict[str,int])*) – starting nexus/editlog offset by layer iden. Defaults to 0 for unspecified layers or if offsdict is None.
- **wait** (*bool*) – whether to pend and stream value until this layer is fini'd

async **updateCronJob**(*iden*, *query*)

Change an existing cron job's query

Parameters
iden (*bytes*) – The iden of the cron job to be changed

async **verifyStormPkgDeps**(*pkgdef*)

viewapi

alias of *ViewApi*

async classmethod **viewctor**(**args*, ***kwargs*)

async **waitStormSvc**(*name*, *timeout=None*)

async **watchAllUserNotifs**(*offs=None*)

class synapse.cortex.**CortexAxonHttpBySha256InvalidV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *CortexAxonMixin*, *AxonHttpBySha256InvalidV1*

class synapse.cortex.**CortexAxonHttpBySha256V1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *CortexAxonMixin*, *AxonHttpBySha256V1*

class synapse.cortex.**CortexAxonHttpDelV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *CortexAxonMixin*, *AxonHttpDelV1*

class synapse.cortex.**CortexAxonHttpHasV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *CortexAxonMixin*, *AxonHttpHasV1*

class synapse.cortex.**CortexAxonHttpUploadV1**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Bases: *CortexAxonMixin*, *AxonHttpUploadV1*

class synapse.cortex.**CortexAxonMixin**

Bases: object

getAxon()

async getAxonInfo()

async prepare()

synapse.cortex.**cmprkey_buid**(*x*)

synapse.cortex.**cmprkey_indx**(*x*)

synapse.cortex.**getTempCortex**(*mods=None*)

Get a proxy to a cortex backed by a temporary directory.

Parameters

mods (*list*) – A list of modules which are loaded into the cortex.

Notes

The cortex and temporary directory are town down on exit. This should only be called from synchronous code.

Returns

Proxy to the cortex.

synapse.cortex.**stormlogger = <Logger synapse.storm (WARNING)>**

A Cortex implements the synapse hypergraph object.

async synapse.cortex.**wrap_liftgenr**(*iden*, *genr*)

10.1.7 synapse.cryotank module

class synapse.cryotank.**CryoApi**

Bases: *CellApi*

The CryoCell API as seen by a telepath proxy.

This is the API to reference for remote CryoCell use.

delete(*name*)

async init(*name*, *conf=None*)

async last(*name*)

async list()

async metrics(*name*, *offs*, *size=None*)

async puts(*name*, *items*)

async rows(*name*, *offs*, *size*)

async slice(*name*, *offs*, *size=None*, *wait=False*, *timeout=None*)

class synapse.cryotank.**CryoCell**

Bases: *Cell*

cellapi

alias of *CryoApi*

async **delete**(*name*)

async **getCellApi**(*link*, *user*, *path*)

Get an instance of the telepath Client object for a given user, link and path.

Parameters

- **link** (*s_link.Link*) – The link object.
- **user** (*s_hive.HiveUser*) – The heavy user object.
- **path** (*str*) – The path requested.

Notes

This defaults to the self.cellapi class. Implementors may override the default class attribute for cellapi to share a different interface.

Returns
The shared object for this cell.

Return type
object

classmethod **getEnvPrefix**()

Get a list of envar prefixes for config resolution.

async **init**(*name*, *conf=None*, *user=None*)

Generate a new CryoTank with a given name or get a reference to an existing CryoTank.

Parameters

- **name** (*str*) – Name of the CryoTank.
- **user** (HiveUser) – The Telepath user.

Returns
A CryoTank instance.

Return type
CryoTank

async **list**(*user=None*)

Get a list of (name, info) tuples for the CryoTanks.

Returns
A list of tufos. user (HiveUser): The Telepath user.

Return type
list

tankapi

alias of *TankApi*

class synapse.cryotank.**CryoTank**

Bases: *Base*

A CryoTank implements a stream of structured data.

iden()

async **info**()

Returns information about the CryoTank instance.

Returns
A dict containing items and metrics indexes.

Return type
dict

last()

Return an (offset, item) tuple for the last element in the tank (or None).

async **metrics**(*offs*, *size=None*)

Yield metrics rows starting at offset.

Parameters

- **offs** (*int*) – The index offset.
- **size** (*int*) – The maximum number of records to yield.

Yields
((int, dict)) – An index offset, info tuple for metrics.

async **puts**(*items*)

Add the structured data from items to the CryoTank.

Parameters
items (*list*) – A list of objects to store in the CryoTank.

Returns
The ending offset of the items or seqn.

Return type
int

async **rows**(*offs*, *size=None*)

Yield a number of raw items from the CryoTank starting at a given offset.

Parameters

- **offs** (*int*) – The index of the desired datum (starts at 0)
- **size** (*int*) – The max number of items to yield.

Yields
((indx, bytes)) – Index and msgpacked bytes.

async **slice**(*offs*, *size=None*, *wait=False*, *timeout=None*)

Yield a number of items from the CryoTank starting at a given offset.

Parameters

- **offs** (*int*) – The index of the desired datum (starts at 0)
- **size** (*int*) – The max number of items to yield.
- **wait** (*bool*) – Once caught up, yield new results in realtime
- **timeout** (*int*) – Max time to wait for a new item.

Yields
((index, object)) – Index and item values.

class synapse.cryotank.**TankApi**

Bases: *CellApi*

async **iden**()

async **metrics**(*offs*, *size=None*)

async **puts**(*items*)

async **slice**(*offs*, *size=None*, *wait=False*, *timeout=None*)

10.1.8 synapse.daemon module

class synapse.daemon.**AsyncGenr**

Bases: *Share*

typename = 'genr'

class synapse.daemon.**Daemon**

Bases: *Base*

async **getSessInfo**()

async **listen**(*url*, ***opts*)

Bind and listen on the given host/port with possible SSL.

Parameters

- **host** (*str*) – A hostname or IP address.
- **port** (*int*) – The TCP port to bind.

async **setReady**(*ready*)

share(*name*, *item*)

Share an object via the telepath protocol.

Parameters

- **name** (*str*) – Name of the shared object
- **item** (*object*) – The object to share over telepath.

class synapse.daemon.**Genr**

Bases: *Share*

typename = 'genr'

class synapse.daemon.**Sess**

Bases: *Base*

getSessItem(*name*)

pack()

popSessItem(*name*)

setSessItem(*name*, *item*)

async synapse.daemon.**t2call**(*link*, *meth*, *args*, *kwargs*)

Call the given `meth(*args, **kwargs)` and handle the response to provide telepath task v2 events to the given link.

10.1.9 synapse.datamodel module

An API to assist with the creation and enforcement of cortex data models.

class synapse.datamodel.**Edge**(*modl*, *edgetype*, *edgeinfo*)

Bases: object

pack()

class synapse.datamodel.**Form**(*modl*, *name*, *info*)

Bases: object

The Form class implements data model logic for a node form.

delProp(*name*)

getFormDef()

getRefsOut()

getStorNode(*form*)

offAdd(*func*)

Unregister a callback for tag addition.

Parameters

- **name** (*str*) – The name of the tag.
- **func** (*function*) – The callback func(node)

onAdd(*func*)

Add a callback for adding this type of node.

The callback is executed after node construction.

Parameters
func (*function*) – A callback func(node)

def func(xact, node):
dostuff()

onDel(*func*)

pack()

prop(*name: str*)

Return a secondary property for this form by relative prop name.

Parameters
name (*str*) – The relative property name.

Returns
The property or None.

Return type
(*synapse.datamodel.Prop*)

setProp(*name*, *prop*)

async **wasAdded**(*node*)

Fire the onAdd() callbacks for node creation.

async **wasDeleted**(*node*)

Fire the onDel() callbacks for node deletion.

class synapse.datamodel.**Model**

Bases: object

The data model used by a Cortex hypergraph.

addBaseType(*item*)

Add a Type instance to the data model.

addDataModels(*mods*)

Add a list of (name, mdef) tuples.

A model definition (mdef) is structured as follows:

```
{
    "ctors":(
        ('name', 'class.path.ctor', {}, {'doc': 'The foo thing.'}),
    ),

    "types":(
        ('name', ('basetype', {typeopts}), {info}),
    ),

    "forms":(
        (formname, (typename, typeopts), {info}, (
            (propname, (typename, typeopts), {info}),
        )),
    ),
    "univs":(
        (propname, (typename, typeopts), {info}),
    )
    "tagprops":(
        (tagpropname, (typename, typeopts), {info}),
    )
    "interfaces":(
        (ifacename, {
            'props': ((propname, (typename, typeopts), {info}),),
            'doc': docstr,
            'interfaces': (ifacename,)
        }),
    )
}
```

Parameters

mods (*list*) – The list of tuples.

Returns

None

addEdge(*edgetype*, *edgeinfo*)

`addForm`(*formname*, *forminfo*, *propdefs*)

`addFormProp`(*formname*, *propname*, *tdef*, *info*)

`addIface`(*name*, *info*)

`addTagProp`(*name*, *tdef*, *info*)

`addType`(*typename*, *basename*, *typeopts*, *typeinfo*)

`addUnivProp`(*name*, *tdef*, *info*)

`delForm`(*formname*)

`delFormProp`(*formname*, *propname*)

`delTagProp`(*name*)

`delType`(*typename*)

`delUnivProp`(*propname*)

`form`(*name*)

`getArrayPropsByType`(*name*)

`getFormsByPrefix`(*prefix*)

`getModelDefs`()

Returns
A list of one model definition compatible with addDataModels that represents the current data model

`getModelDict`()

`getProps`()

`getPropsByType`(*name*)

`getTagProp`(*name*)

`getTypeClone`(*typedef*)

`prop`(*name*)

`reqFormsByLook`(*name*, *extra=None*)

`reqFormsByPrefix`(*prefix*, *extra=None*)

`reqPropsByLook`(*name*, *extra=None*)

`tagprop`(*name*)

`type`(*name*)

Return a synapse.lib.types.Type by name.

`univ`(*name*)

class synapse.datamodel.**Prop**(*modl*, *form*, *name*, *typedef*, *info*)

Bases: `object`

The Prop class represents a property defined within the data model.

getCompOffs()

Return the offset of this field within the compound primary prop or None.

getPropDef()

getStorNode(*form*)

onDel(*func*)

Add a callback for deleting this property.

The callback is executed after the property is deleted.

Parameters
func (*function*) – A prop del callback.

The callback is called within the current transaction, with the node, and the old property value (or None).

def func(node, oldv):
dostuff()

onSet(*func*)

Add a callback for setting this property.

The callback is executed after the property is set.

Parameters
func (*function*) – A prop set callback.

The callback is called within the current transaction, with the node, and the old property value (or None).

def func(node, oldv):
dostuff()

pack()

async wasDel(*node*, *oldv*)

async wasSet(*node*, *oldv*)

Fire the onset() handlers for this property.

Parameters

- **node** (`synapse.lib.node.Node`) – The node whose property was set.
- **oldv** (*obj*) – The previous value of the property.

class synapse.datamodel.**TagProp**(*model*, *name*, *tdef*, *info*)

Bases: `object`

getStorNode(*form*)

getTagPropDef()

pack()

10.1.10 synapse.exc module

Exceptions used by synapse, all inheriting from SynErr

exception synapse.exc.**AuthDeny**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BackupAlreadyRunning**(**args*, ***info*)

Bases: *SynErr*

Only one backup may be running at a time

exception synapse.exc.**BadArg**(**args*, ***info*)

Bases: *SynErr*

Improper function arguments

exception synapse.exc.**BadCast**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadCertBytes**(**args*, ***info*)

Bases: *SynErr*

Raised by certdir when the certificate fails to load.

exception synapse.exc.**BadCertHost**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadCertVerify**(**args*, ***info*)

Bases: *SynErr*

Raised by certdir when there is a failure to verify a certificate context.

exception synapse.exc.**BadCmdName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadCmprType**(**args*, ***info*)

Bases: *SynErr*

Attempt to compare two incomparable values

exception synapse.exc.**BadCmprValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadConfValu**(**args*, ***info*)

Bases: *SynErr*

The configuration value provided is not valid.

This should contain the config name, valu and mesg.

exception synapse.exc.**BadCoreStore**(**args*, ***info*)

Bases: *SynErr*

The storage layer has encountered an error

exception synapse.exc.**BadCtorType**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadDataValu**(**args*, ***info*)

Bases: *SynErr*

Cannot process the data as intended.

exception synapse.exc.**BadEccExchange**(**args*, ***info*)

Bases: *CryptoErr*

Raised when there is an issue doing a ECC Key Exchange

exception synapse.exc.**BadFileExt**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadFormDef**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadHivePath**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadIndxValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadJsonText**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadLiftValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadMesgFormat**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadMesgVers**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadOperArg**(**args*, ***info*)

Bases: *SynErr*

Improper storm function arguments

exception synapse.exc.**BadOptValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadPkgDef**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadPropDef**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadState**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadStorageVersion**(**args*, ***info*)

Bases: *SynErr*

Stored persistent data is incompatible with running software

exception synapse.exc.**BadSyntax**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadTag**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadTime**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadTypeDef**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadTypeValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadUrl**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**BadVersion**(**args*, ***info*)

Bases: *SynErr*

Generic Bad Version exception.

exception synapse.exc.**CantDelCmd**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelForm**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelNode**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelProp**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelType**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelUniv**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantDelView**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantMergeView**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CantRevLayer**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**CliFini**(**args*, ***info*)

Bases: *SynErr*

Raised when the CLI is to exit.

exception synapse.exc.**CryptoErr**(**args*, ***info*)

Bases: *SynErr*

Raised when there is a synapse.lib.crypto error.

exception synapse.exc.**DataAlreadyExists**(**args*, ***info*)

Bases: *SynErr*

Cannot copy data to a location that already contains data

exception synapse.exc.**DbOutOfSpace**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DmonSpawn**(**args*, ***info*)

Bases: *SynErr*

Raised by a dispatched telepath method that has answered the call using a spawned process. (control flow that is compatible with aborting standard calls, generators, and async generators).

exception synapse.exc.**DupFileName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupFormName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupIden**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupIndx**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupPropName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupRoleName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupStormSvc**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupTagPropName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**DupUserName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**FatalErr**(**args*, ***info*)

Bases: *SynErr*

Raised when a fatal error has occurred which an application cannot recover from.

exception synapse.exc.**FeatureNotSupported**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**FileExists**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**HitLimit**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**InconsistentStorage**(**args*, ***info*)

Bases: *SynErr*

Stored persistent data is inconsistent

exception synapse.exc.**IsDeprLocked**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**IsFini**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**IsReadOnly**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**IsRuntForm**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**LayerInUse**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**LinkBadCert**(**args*, ***info*)

Bases: *LinkErr*

exception synapse.exc.**LinkErr**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**LinkShutDown**(**args*, ***info*)

Bases: *LinkErr*

exception synapse.exc.**LmdbLock**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**LowSpace**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**ModAlreadyLoaded**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**MustBeJsonSafe**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NeedConfValu**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoCertKey**(**args*, ***info*)

Bases: *SynErr*

Raised when a Cert object requires a RSA Private Key to perform an operation and the key is not present.

exception synapse.exc.**NoSuchAbrv**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchAct**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchAuthGate**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchCert**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchCmd**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchCmpr**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchCond**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchCtor**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchDecoder**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchDir**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchDyn**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchEncoder**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchFile**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchForm**(**args*, ***info*)

Bases: *SynErr*

classmethod init(*name*, *mesg=None*)

exception synapse.exc.**NoSuchFunc**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchIden**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchImpl**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchIndx**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchLayer**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchLift**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchMeth**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchName**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchObj**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchOpt**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchPath**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchPivot**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchPkg**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchProp**(**args*, ***info*)

Bases: *SynErr*

classmethod init(*name*, *mesg=None*)

exception synapse.exc.**NoSuchRole**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchStormSvc**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchTagProp**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchType**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchUniv**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchUser**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchVar**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NoSuchView**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NotANumberCompared**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NotMsgpackSafe**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**NotReady**(**args*, ***info*)

Bases: *Retry*

exception synapse.exc.**ParserExit**(**args*, ***info*)

Bases: *SynErr*

Raised by synapse.lib.cmd.Parser on Parser exit()

exception synapse.exc.**PathExists**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**ReadOnlyLayer**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**ReadOnlyProp**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**RecursionLimitHit**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**Retry**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**SchemaViolation**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**SlabAlreadyOpen**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**SlabInUse**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**SpawnExit**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**StepTimeout**(**args*, ***info*)

Bases: *SynErr*

Raised when a TestStep.wait() call times out.

exception synapse.exc.**StormPkgConflicts**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**StormPkgRequires**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**StormRaise**(**args*, ***info*)

Bases: *SynErr*

This represents a user provided exception inside of a Storm runtime. It requires a errname key.

exception synapse.exc.**StormRuntimeError**(**args*, ***info*)

Bases: *SynErr*

exception synapse.exc.**StormVarListError**(**args*, ***info*)

Bases: *StormRuntimeError*

exception synapse.exc.**SynErr**(**args*, ***info*)

Bases: Exception

get(*name*, *defv=None*)

Return a value from the errinfo dict.

Example

try:
foothing()

except SynErr as e:
blah = e.get('blah')

items()

set(*name*, *valu*)
Set a value in the errinfo dict.

setdefault(*name*, *valu*)
Set a value in errinfo dict if it is not already set.

exception synapse.exc.**TimeOut**(**args*, ***info*)
Bases: *SynErr*

10.1.11 synapse.glob module

synapse.glob.**iAmLoop**()

synapse.glob.**initloop**()

synapse.glob.**setGreedCoro**(*loop: AbstractEventLoop*)

synapse.glob.**sync**(*coro*, *timeout=None*)
Schedule a coroutine to run on the global loop and return it's result.

Parameters
coro (*coroutine*) – The coroutine instance.

Notes

This API is thread safe and should only be called by non-loop threads.

synapse.glob.**synchelp**(*f*)
The synchelp decorator allows the transparent execution of a coroutine using the global loop from a thread other than the event loop. In both use cases, the actual work is done by the global event loop.

Examples

Use as a decorator:

```
@s_glob.synchelp
async def stuff(x, y):
    await dostuff()
```

Calling the stuff function as regular async code using the standard await syntax:

```
valu = await stuff(x, y)
```

Calling the stuff function as regular sync code outside of the event loop thread:

```
valu = stuff(x, y)
```

10.1.12 synapse.mindmeld module

10.1.13 synapse.telepath module

An RMI framework for synapse.

class synapse.telepath.**Aware**

Bases: object

The telepath.Aware mixin allows shared objects to handle individual links managed by the Daemon.

async getTeleApi(*link*, *mesg*, *path*)

Return a shared object for this link. :param link: A network link. :type link: synapse.lib.link.Link :param mesg: The tele:syn handshake message. :type mesg: (str,dict)

onTeleShare(*dmon*, *name*)

class synapse.telepath.**Client**

Bases: *Base*

A Telepath client object which reconnects and allows waiting for link up.

Notes

The conf data allows changing parameters such as timeouts, retry period, and link pool size. The default conf data can be seen below:

```
conf = {
    'timeout': 10,
    'retrysleep': 0.2,
    'link_poolsize': 4,
}
```

async offlink(*func*)

async onlink(*func*)

async proxy(*timeout=10*)

async task(*todo*, *name=None*)

async waitready(*timeout=10*)

class synapse.telepath.**ClientV2**

Bases: *Base*

A telepath client which:

- connects to multiple services
- distributes API calls across them
- receives topology updates from AHA

NOTE: This must co-exist with Client until we eliminate uses that
attempt to call telepath APIs directly from the Client rather than awaiting a proxy()

`getNextBootUrl()`

`async proxy(timeout=None)`

`size()`

`async waitready(timeout=None)`

`class synapse.telepath.Genr`

Bases: *Share*

`class synapse.telepath.GenrIter(proxy, todo, share)`

Bases: `object`

An object to help delay a telepath call until iteration.

`async list()`

`class synapse.telepath.GenrMethod(proxy, name, share=None)`

Bases: *Method*

`class synapse.telepath.Method(proxy, name, share=None)`

Bases: `object`

The telepath Method is used to provide proxy method calls.

`class synapse.telepath.Pipeline`

Bases: *Base*

`class synapse.telepath.Proxy`

Bases: *Base*

A telepath Proxy is used to call remote APIs on a shared object.

Example

import synapse.telepath as s_telepath

open the "foo" object shared in a dmon on localhost:3344

async def doFooThing():

proxy = await s_telepath.openurl('tcp://127.0.0.1:3344/foo')

valu = await proxy.getFooValu(x, y)

The proxy (and openurl function) may also be used from sync code:

proxy = s_telepath.openurl('tcp://127.0.0.1:3344/foo')

valu = proxy.getFooValu(x, y)

`async call(methname, *args, **kwargs)`

Call a remote method by name.

Parameters

- **methname** (*str*) – The name of the remote method.
- ***args** – Arguments to the method call.

- `**kwargs` – Keyword arguments to the method call.

Most use cases will likely use the proxy methods directly:

The following two are effectively the same:

valu = proxy.getFooBar(x, y) valu = proxy.call('getFooBar', x, y)

async **getPipeline**(*genr*, *name=None*)

Construct a proxy API call pipeline in order to make multiple telepath API calls while minimizing round trips.

Parameters

- **genr** (*async generator*) – An async generator that yields todo tuples.
- **name** (*str*) – The name of the shared object on the daemon.

Example

def genr():
yield s_common.todo('getFooByBar', 10) yield s_common.todo('getFooByBar', 20)

for retn in proxy.getPipeline(genr()):
valu = s_common.result(retn)

async **getPoolLink**()

async **handshake**(*auth=None*)

async **task**(*todo*, *name=None*)

async **taskv2**(*todo*, *name=None*)

class synapse.telepath.**Share**

Bases: *Base*

The telepath client side of a dynamically shared object.

class synapse.telepath.**Task**

Bases: `object`

A telepath Task is used to internally track calls/responses.

reply(*retn*)

async **result**()

class synapse.telepath.**TeleSSLObject**(**args*, ***kwargs*)

Bases: `SSLObject`

do_handshake()

Start the SSL/TLS handshake.

async synapse.telepath.**addAhaUrl**(*url*)

Add (incref) an aha registry URL.

NOTE: You may also add a list of redundant URLs.

synapse.telepath.**alias**(*name*)

Resolve a telepath alias via ~/.syn/aliases.yaml

Parameters
`name` (*str*) – Name of the alias to resolve.

Notes

An exact match against the aliases will always be returned first. If no exact match is found and the name contains a '/' in it, the value before the slash is looked up and the remainder of the path is joined to any result. This is done to support dynamic Telepath share names.

Returns
The url string, if present in the alias. None will be returned if there are no matches.

Return type
str

synapse.telepath.**chopurl**(*url*, ***opts*)

async synapse.telepath.**delAhaUrl**(*url*)

Remove (decref) an aha registry URL.

NOTE: You may also remove a list of redundant URLs.

async synapse.telepath.**getAhaProxy**(*urlinfo*)

Return a telepath proxy by looking up a host from an AHA registry.

synapse.telepath.**loadTeleCell**(*dirn*)

async synapse.telepath.**loadTeleEnv**(*path*)

synapse.telepath.**mergeAhaInfo**(*info0*, *info1*)

synapse.telepath.**modurl**(*url*, ***info*)

async synapse.telepath.**open**(*url*, *onlink=None*)

Open a new telepath ClientV2 object based on the given URL.

Parameters

- `url` (*str*) – The URL to connect to.
- `onlink` – An optional async callback function to run when connections are made.

Notes

The onlink callback function has the call signature `(proxy, urlinfo)`. The proxy is the Telepath Proxy object. The urlinfo is the parsed URL information used to create the proxy object. The urlinfo structure may change between versions of Synapse.

Returns
A ClientV2 object.

Return type
ClientV2

async synapse.telepath.**openinfo**(*info*) `synapse.telepath.`**`withTeleEnv`**`()`

`synapse.telepath.`**`zipurl`**`(`*info*`)`

Reconstruct a URL string from a parsed telepath info dict.

CHAPTER

ELEVEN

SYNAPSE HTTP/REST API

Many components within the Synapse ecosystem provide HTTP/REST APIs to provide a portable interface. Some of these APIs are RESTful, while other (streaming data) APIs are technically not.

11.1 HTTP/REST API Conventions

All Synapse RESTful APIs use HTTP GET/POST methods to retrieve and modify data. All POST requests expect a JSON body. Each RESTful API call will return a result wrapper dictionary with one of two conventions.

For a successful API call:

```
{"status": "ok", "result": "some api result here"}
```

or for an unsuccessful API call:

```
{"status": "err": "code": "ErrCodeString", "mesg": "A human friendly message."}
```

Streaming HTTP API endpoints, such as the interface provided to retrieve nodes from a Synapse Cortex, provide JSON results via HTTP chunked encoding where each chunk is a single result.

The client example code in these docs is given with the Python "aiohttp" and "requests" modules. They should be enough to understand the basic operation of the APIs.

For additional examples, see the code examples at HTTPAPI Examples.

11.2 Authentication

Most Synapse HTTP APIs require an authenticated user. HTTP API endpoints requiring authentication may be accessed using either HTTP Basic authentication via the HTTP "Authorization" header, using an API Key with the "X-API-KEY" header, or as part of an authenticated session.

11.2.1 API Key Support

A Cortex user can create their own API key via Storm. The following is an example of generating a user API key:

```
storm> ($key, $info)= $lib.auth.users.byname($lib.user.name()).genApiKey('Test␣
↪Key') $lib.print($key)
XauBgBIUKgWJEm7VyvkmcuaGZbIl6M2nmueWjRtnYtA=
```

This API Key can then be used to make HTTP API calls. The following example shows the use of `curl` and `jq` to make a Storm call with the API key and then format the response:

```
$ curl -k -s -H "X-API-KEY: XauBgBIUKgWJEm7VyvkmcuaGZbIl6M2nmueWjRtnYtA=" \
--data '{"query": "return($lib.user.name())"}' \
https://localhost:4443/api/v1/storm/call | jq

{
  "status": "ok",
  "result": "root"
}
```

11.2.2 /api/v1/login

The login API endpoint may be used to create an authenticated session. To create and use an authenticated session, the HTTP client library must support cookies. This session may then be used to call other HTTP API endpoints as the authenticated user. This expects a `user` and `passwd` provided in the body of a `POST` request. The reusable session cookie is returned in a `Set-Cookie` header.

Both of the Python examples use session managers which manage the session cookie automatically.

```python
import aiohttp

async def logInExample(ssl=False):

    async with aiohttp.ClientSession() as sess:

        info = {'user': 'visi', 'passwd': 'secret'}
        async with sess.post('https://localhost:4443/api/v1/login', json=info, ssl=ssl)␣
↪as resp:
            item = await resp.json()
            if item.get('status') != 'ok':
                code = item.get('code')
                mesg = item.get('mesg')
                raise Exception(f'Login error ({code}): {mesg}')

            # we are now clear to make additional HTTP API calls using sess
```

```python
import requests

def logInExample(ssl=False):

    sess = requests.session()

    url = 'https://localhost:4443/api/v1/login'
```

(continues on next page)

(continued from previous page)

```
    info = {'user': 'visi', 'passwd': 'secret'}
    resp = sess.post(url, json=info, verify=ssl)
    item = resp.json()

    if item.get('status') != 'ok':
        code = item.get('code')
        mesg = item.get('mesg')
        raise Exception(f'Login error ({code}): {mesg}')

    # we are now clear to make additional HTTP API calls using sess
```

11.2.3 /api/v1/logout

The logout API endpoint may be used to end an authenticated session. This invalidates the session, and any further requests to authenticated endpoints will fail on authentication failed errors.

Both of the Python examples use session managers which manage the session cookie automatically.

```
import aiohttp

def logoutExample(sess, ssl):
    url = 'https://localhost:4443/api/v1/logout'
    resp = sess.get(url, ssl=ssl)
    item = resp.json()
    if item.get('status') != 'ok':
        code = item.get('code')
        mesg = item.get('mesg')
        raise Exception(f'Logout error ({code}): {mesg}')
```

```
import requests

def logoutExample(sess, ssl):
    url = 'https://localhost:4443/api/v1/logout'
    resp = sess.get(url, verify=ssl)
    item = resp.json()
    if item.get('status') != 'ok':
        code = item.get('code')
        mesg = item.get('mesg')
        raise Exception(f'Logout error ({code}): {mesg}')
```

11.2.4 /api/v1/active

Method

GET

This is an unauthenticated API that returns the leader status of Cell.

Returns

A dictionary with the `active` key set to True or False.

11.2.5 /api/v1/auth/users

Method
GET

Returns
A list of dictionaries, each of which represents a user on the system.

11.2.6 /api/v1/auth/roles

Method
GET

Returns
A list of dictionaries, each of which represents a role on the system.

11.2.7 /api/v1/auth/adduser

Method
POST

This API endpoint allows the caller to add a user to the system.

Input
This API expects the following JSON body:

```
{ "name": "myuser" }
```

Any additional "user dictionary" fields (other than "iden") may be specified.

Returns
The newly created user dictionary.

11.2.8 /api/v1/auth/addrole

Method
POST

This API endpoint allows the caller to add a role to the system.

Input
This API expects the following JSON body:

```
{ "name": "myrole" }
```

Any additional "role dictionary" fields (other than "iden") may be specified.

Returns
The newly created role dictionary.

11.2.9 /api/v1/auth/delrole

Method

POST

This API endpoint allows the caller to delete a role from the system.

Input

This API expects the following JSON body:

```
{ "name": "myrole" }
```

Returns

null

11.2.10 /api/v1/auth/user/<id>

Method

POST

This API allows the caller to modify specified elements of a user dictionary.

Input

This API expects a JSON dictionary containing any updated values for the user.

Returns

The updated user dictionary.

Method

GET

This API allows the caller to retrieve a user dictionary.

Returns

A user dictionary.

11.2.11 /api/v1/auth/password/<id>

Method

POST

This API allows the caller to change a user's password. The authenticated user must either be an admin or the user whose password is being changed.

Input

This API expects a JSON dictionary containing the key `passwd` with the new password string.

Returns

The updated user dictionary.

11.2.12 /api/v1/auth/role/<id>

Method

POST

This API allows the caller to modify specified elements of a role dictionary.

Input

This API expects a dictionary containing any updated values for the role.

Returns

The updated role dictionary.

Method

GET

This API allows the caller to retrieve a role dictionary.

Returns

A role dictionary.

11.2.13 /api/v1/auth/grant

Method

POST

This API allows the caller to grant a role to a given user.

Input

This API expects the following JSON body:

```
{
    "user": "<id>",
    "role": "<id>"
}
```

Returns

The updated user dictionary.

11.2.14 /api/v1/auth/revoke

Method

POST

This API allows the caller to revoke a role which was previously granted to a user.

Input

This API expects the following JSON body:

```
{
    "user": "<id>",
    "role": "<id>"
}
```

Returns

The updated user dictionary.

11.3 Cortex

A Synapse Cortex implements an HTTP API for interacting with the hypergraph and data model. Some of the provided APIs are pure REST APIs for simple data model operations and single/simple node modification. However, many of the HTTP APIs provided by the Cortex are streaming APIs which use HTTP chunked encoding to deliver a stream of results as they become available.

The Cortex also implements the *Axon* HTTP API. Permissions are checked within the Cortex, and then the request is executed on the Axon.

11.3.1 /api/v1/feed

The Cortex feed API endpoint allows the caller to add nodes in bulk.

Method
POST

Input
The API expects the following JSON body:

```
{
    "items": [ <node>, ... ],
    # and optionally...
    "view": <iden>,
}
```

Each <node> is expected to be in packed tuple form:

```
[ [<formname>, <formvalu>], {...} ]
```

Returns
The API returns `{"status": "ok", "result": null}` on success and any failures are returned using the previously mentioned REST API convention.

11.3.2 /api/v1/storm

The Storm API endpoint allows the caller to execute a Storm query on the Cortex and stream back the messages generated during the Storm runtime execution. In addition to returning nodes, these messages include events for node edits, tool console output, etc. This streaming API has back-pressure, and will handle streaming millions of results as the reader consumes them. For more information about Storm APIs, including opts behavior, see *Storm API Guide*.

Method
GET

Input
The API expects the following JSON body:

```
{
    "query": "a storm query here",

    # optional
    "opts": {
        ...
    }
```

(continues on next page)

(continued from previous page)

```
    # optional
    "stream": "jsonlines"
}
```

Returns

The API returns a series of messages generated by the Storm runtime. Each message is returned as an HTTP chunk, allowing readers to consume the resulting messages as a stream.

The `stream` argument to the body modifies how the results are streamed back. Currently this optional argument can be set to `jsonlines` to get newline separated JSON data.

Examples

The following two examples show querying the `api/v1/storm` endpoint and receiving multiple message types.

aiohttp example:

```
import json
import pprint

# Assumes sess is an aiohttp client session that has previously logged in

query = '.created $lib.print($node.repr(".created")) | limit 3'
data = {'query': query, 'opts': {'repr': True}}
url = 'https://localhost:4443/api/v1/storm'

async with sess.get(url, json=data) as resp:
    async for byts, x in resp.content.iter_chunks():

        if not byts:
            break

        mesg = json.loads(byts)
        pprint.pprint(mesg)
```

requests example:

```
import json
import pprint
# Assumes sess is an requests client session that has previously logged in

query = '.created $lib.print($node.repr(".created")) | limit 3'
data = {'query': query, 'opts': {'repr': True}}
url = 'https://localhost:4443/api/v1/storm'

resp = sess.get(url, json=data, stream=True)
for chunk in resp.iter_content(chunk_size=None, decode_unicode=True):
    mesg = json.loads(chunk)
    pprint.pprint(mesg)
```

When working with these APIs across proxies, we have experienced issues with NGINX interfering with the chunked encoding. This may require more careful message reconstruction. The following shows using aiohttp to do that message reconstruction.

```
import json
import pprint
# Assumes sess is an requests client session that has previously logged in

query = '.created $lib.print($node.repr(".created")) | limit 3'
data = {'query': query, 'opts': {'repr': True}}
url = 'https://localhost:4443/api/v1/storm'

async with sess.get(url, json=data) as resp:

    buf = b""

    async for byts, chunkend in resp.content.iter_chunks():

        if not byts:
            break

        buf += byts
        if not chunkend:
            continue

        mesg = json.loads(buf)
        buf = b""

        pprint.pprint(buf)
```

11.3.3 /api/v1/storm/call

The Storm Call API endpoint allows the caller to execute a Storm query on the Cortex and get a single return value back from the runtime. This is analogous to using the `callStorm()` Telepath API. This expects to return a value from the Storm query using the Storm `return( )` syntax. For more information about Storm APIs, including opts behavior, see *Storm API Guide*.

Method

GET

Input

The API expects the following JSON body:

```
{
    "query": "a storm query here",

    # optional
    "opts": {
        ...
    }
}
```

Returns

The API returns `{"status": "ok", "result": return_value}` on success and any failures are returned using the previously mentioned REST API convention.

Examples

The following two examples show querying the `api/v1/storm/call` endpoint and receiving a return value.

aiohttp example:

```
import pprint

# Assumes sess is an aiohttp client session that has previously logged in

query = '$foo = $lib.str.format("hello {valu}", valu="world") return ($foo)'
data = {'query': query}
url = 'https://localhost:4443/api/v1/storm/call'

async with sess.get(url, json=data) as resp:
    info = await resp.json()
    pprint.pprint(info)
```

requests example:

```
import pprint
# Assumes sess is an requests client session that has previously logged in

query = '$foo = $lib.str.format("hello {valu}", valu="world") return ($foo)'
data = {'query': query}
url = 'https://localhost:4443/api/v1/storm/call'

resp = sess.get(url, json=data)
info = resp.json()
pprint.pprint(info)
```

11.3.4 /api/v1/storm/nodes

Warning: This API is deprecated in Synapse `v2.110.0` and will be removed in a future version.

The Storm nodes API endpoint allows the caller to execute a Storm query on the Cortex and stream back the resulting nodes. This streaming API has back-pressure, and will handle streaming millions of results as the reader consumes them.

Method

GET

Input

See /api/v1/storm for expected JSON body input.

Returns

The API returns the resulting nodes from the input Storm query. Each node is returned as an HTTP chunk, allowing readers to consume the resulting nodes as a stream.

Each serialized node will have the following structure:

```
[
    [<form>, <valu>],       # The [ typename, typevalue ] definition of the␣
↪node.
    {
```

(continues on next page)

(continued from previous page)

```
        "iden": <hash>,      # A stable identifier for the node.
        "tags": {},          # The tags on the node.
        "props": {},         # The node's secondary properties.

        # optionally (if query opts included {"repr": True}
        "reprs": {}          # Presentation values for props which need it.
    }
]
```

The `stream` argument, documented in the /api/v1/storm endpoint, modifies how the nodes are streamed back. Currently this optional argument can be set to `jsonlines` to get newline separated JSON data.

11.3.5 /api/v1/storm/export

The Storm export API endpoint allows the caller to execute a Storm query on the Cortex and export the resulting nodes in msgpack format such that they can be directly ingested with the `syn.nodes` feed function.

Method
GET

Input
See /api/v1/storm for expected JSON body input.

Returns
The API returns the resulting nodes from the input Storm query. This API yields nodes after an initial complete lift in order to limit exported edges.

Each exported node will be in msgpack format.

There is no Content-Length header returned, since the API cannot predict the volume of data a given query may produce.

11.3.6 /api/v1/model

Method
GET

This API allows the caller to retrieve the current Cortex data model.

Input
The API takes no input.

Returns
The API returns the model in a dictionary, including the types, forms and tagprops. Secondary property information is also included for each form:

```
{
    "types": {
        ...  # dictionary of type definitions
    },
    "forms": {
        ...  # dictionary of form definitions, including secondary properties
    },
    "tagprops": {
```

(continues on next page)

(continued from previous page)

```
        ...  # dictionary of tag property definitions
    }
}
```

11.3.7 /api/v1/model/norm

Method

GET, POST

This API allows the caller to normalize a value based on the Cortex data model. This may be called via a GET or POST requests.

Input

The API expects the following JSON body:

```
{
    "prop": "prop:name:here",
    "value": <value>,
}
```

Returns

The API returns the normalized value as well as any parsed subfields or type specific info:

```
{
    "norm": <value>,
    "info": {
        "subs": {},
        ...
    }
}
```

11.3.8 /api/v1/storm/vars/get

Method

GET

This API allows the caller to retrieve a storm global variable.

Input

The API expects the following JSON body:

```
{
    "name": "varnamehere",
    "default": null,
}
```

Returns

The API returns the global variable value or the specified default using the REST API convention described earlier.

11.3.9 /api/v1/storm/vars/set

Method

POST

This API allows the caller to set a storm global variable.

Input

The API expects the following JSON body:

```
{
    "name": "varnamehere",
    "value": <value>,
}
```

Returns

The API returns *true* using the REST API convention described earlier.

11.3.10 /api/v1/storm/vars/pop

Method

POST

This API allows the caller to pop/delete a storm global variable.

Input

The API expects the following JSON body:

```
{
    "name": "varnamehere",
    "default": <value>,
}
```

Returns

The API returns the current value of the variable or default using the REST API convention described earlier.

11.3.11 /api/v1/core/info

Method

GET

This API allows the caller to retrieve the current Cortex version, data model definitions, and Storm information.

Input

The API takes no input.

Returns

The API returns the model in a dictionary, including the types, forms and tagprops. Secondary property information is also included for each form:

```
{
    "version": [ <major>, <minor>, <patch> ], # Version tuple
    "modeldict": {
        ...  # dictionary of model definitions
```

(continues on next page)

(continued from previous page)

```
    },
    "stormdocs": {
        "libraries": [
            ... # list of information about Storm libraries.
        ],
        "types": [
            ... # list of information about Storm types.
        ]
    }
}
```

11.3.12 /api/ext/*

This API endpoint is used as the Base URL for Extended HTTP API endpoints which are user defined. See *Extended HTTP API* for additional information about this endpoint.

11.4 Aha

A Synapse Aha service implements an HTTP API for assisting with devops.

11.4.1 /api/v1/aha/provision/service

Method

POST

This API allows the caller to generate an AHA provisioning URL.

Input

The API expects the following JSON body:

```
{
    "name": " ... name of the service being provisioned",
    "provinfo": {
        "dmon:port": # optional integer, default Telepath listening port.
        "https:port": # optional integer, default HTTPS listening port.
        "mirror": # optional string, service to Mirror.
        "conf": {
            ... # optional, default service configuration values.
        }
    }
}
```

Returns

The API returns the following provisioning information. The data is returned using the REST API convention described earlier:

```
{
    "url": "< the AHA provisioning URL >",
}
```

11.4.2 /api/v1/aha/services

Method

GET

This API allows the caller to get a list of all the registered services.

Input

The API accepts the following **optional** JSON body:

```
{
    "network": " ... name of the aha network to list",
}
```

Returns

The API returns the following provisioning information. The data is returned using the REST API convention described earlier:

```
[
    {
        "name": "< the full service name >",
        "svcname": "< service name part >",
        "svcnetw": "< service network part >",
        "svcinfo": {
            "run": "< runtime service identifier >",
            "iden": "< persistent service identifier >",
            "leader": "< service leader name >",
            "urlinfo": {
                "scheme": "< listening scheme >",
                "port": listening port,
                "path": "< listening path >",
                "host": "< listening IP address >"
            },
            "ready": < boolean indicating the service is either an active␣
↪leader or in the realtime change event window >,
            "online": < runtime aha identifier if the service is connected >
        }
    },
    ...
]
```

11.5 Axon

A Synapse Axon implements an HTTP API for uploading and downloading files. The HTTP APIs use HTTP chunked encoding for handling large files.

11.5.1 /api/v1/axon/files/del

This API allows the caller to delete multiple files from the Axon by the SHA-256.

Method
POST

Input
The API expects the following JSON body:

```
{
    "sha256s": [<sha256>, ...],
}
```

Returns
The API returns an array of SHA-256 and boolean values representing whether each was found in the Axon and deleted. The array is returned using the REST API convention described earlier.

11.5.2 /api/v1/axon/files/put

This API allows the caller to upload and save a file to the Axon. This may be called via a PUT or POST request.

Method
PUT, POST

Input
The API expects a stream of byte chunks.

Returns
On successful upload, or if the file already existed, the API returns information about the file:

```
{
  "md5": "<the md5sum value of the uploaded bytes>",
  "sha1": "<the sha1 value of the uploaded bytes>",
  "sha256": "<the sha256 value of the uploaded bytes>",
  "sha512": "<the sha512 value of the uploaded bytes>",
  "size": <the size of the uploaded bytes>
}
```

11.5.3 /api/v1/axon/files/has/sha256/<SHA-256>

This API allows the caller to check if a file exists in the Axon as identified by the SHA-256.

Method
GET

Returns
True if the file exists; False if the file does not exist.

11.5.4 /api/v1/axon/files/by/sha256/<SHA-256>

This API allows the caller to retrieve or remove a file from the Axon as identified by the SHA-256. If the file does not exist a 404 will be returned.

Method
GET

Returns
If the file exists a stream of byte chunks will be returned to the caller. A `Range` header with a single `bytes` value can be provided to get a subset of a file.

Method
HEAD

Returns
If the file exists, the `Content-Length` header will be set for the size of the file. If a `Range` header with a single `bytes` value is provided, the `Content-Length` header will describe the size of the range, and the `Content-Range` header will also be set to describe the range of the requested bytes.

Method
DELETE

Returns
Boolean via the REST API convention described earlier. If the file is not found an error is returned.

CHAPTER

TWELVE

SYNAPSE DATA MODEL

This contains documentation for Synapse Data Model, including the data model deprecation policy.

The current sections are:

12.1 Synapse Data Model - Types

12.1.1 Base Types

Base types are defined via Python classes.

array

A typed array which indexes each field. It is implemented by the following class: `synapse.lib.types.Array`.

The base type `array` has the following default options set:

- type: `int`

bool

The base boolean type. It is implemented by the following class: `synapse.lib.types.Bool`.

comp

The base type for compound node fields. It is implemented by the following class: `synapse.lib.types.Comp`.

cvss:v2

A CVSS v2 vector string. It is implemented by the following class: `synapse.models.risk.CvssV2`.

An example of `cvss:v2`:

- `(AV:L/AC:L/Au:M/C:P/I:C/A:N)`

cvss:v3

A CVSS v3.x vector string. It is implemented by the following class: `synapse.models.risk.CvssV3`.

An example of `cvss:v3`:

- `AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L`

data

Arbitrary json compatible data. It is implemented by the following class: `synapse.lib.types.Data`.

duration

A duration value. It is implemented by the following class: `synapse.lib.types.Duration`.

The base type `duration` has the following default options set:

- signed: `False`

edge

An digraph edge base type. It is implemented by the following class: `synapse.lib.types.Edge`.

file:base

A file name with no path. It is implemented by the following class: `synapse.models.files.FileBase`.

An example of `file:base`:

- `woot.exe`

file:bytes

The file bytes type with SHA256 based primary property. It is implemented by the following class: `synapse.models.files.FileBytes`.

file:path

A normalized file path. It is implemented by the following class: `synapse.models.files.FilePath`.

An example of `file:path`:

- `c:/windows/system32/calc.exe`

float

The base floating point type. It is implemented by the following class: `synapse.lib.types.Float`.

The base type `float` has the following default options set:

- fmt: `%f`
- min: `None`
- minisvalid: `True`
- max: `None`
- maxisvalid: `True`

geo:area

A geographic area (base unit is square mm). It is implemented by the following class: `synapse.models.geospace.Area`.

An example of `geo:area`:

- `10 sq.km`

geo:dist

A geographic distance (base unit is mm). It is implemented by the following class: `synapse.models.geospace.Dist`.

An example of `geo:dist`:

- `10 km`

geo:latlong

A Lat/Long string specifying a point on Earth. It is implemented by the following class: `synapse.models.geospace.LatLong`.

An example of `geo:latlong`:

- `-12.45,56.78`

guid

The base GUID type. It is implemented by the following class: `synapse.lib.types.Guid`.

hex

The base hex type. It is implemented by the following class: `synapse.lib.types.Hex`.

The base type `hex` has the following default options set:

- size: `0`
- zeropad: `0`

hugenum

A potentially huge/tiny number. [x] <= 730750818665451459101842 with a fractional precision of 24 decimal digits. It is implemented by the following class: `synapse.lib.types.HugeNum`.

The base type `hugenum` has the following default options set:

- units: `None`
- modulo: `None`

inet:addr

A network layer URL-like format to represent tcp/udp/icmp clients and servers. It is implemented by the following class: `synapse.models.inet.Addr`.

An example of `inet:addr`:

- `tcp://1.2.3.4:80`

inet:cidr4

An IPv4 address block in Classless Inter-Domain Routing (CIDR) notation. It is implemented by the following class: `synapse.models.inet.Cidr4`.

An example of `inet:cidr4`:

- `1.2.3.0/24`

inet:cidr6

An IPv6 address block in Classless Inter-Domain Routing (CIDR) notation. It is implemented by the following class: `synapse.models.inet.Cidr6`.

An example of `inet:cidr6`:

- `2001:db8::/101`

inet:dns:name

A DNS query name string. Likely an FQDN but not always. It is implemented by the following class: `synapse.models.dns.DnsName`.

An example of `inet:dns:name`:

- `vertex.link`

inet:email

An e-mail address. It is implemented by the following class: `synapse.models.inet.Email`.

inet:fqdn

A Fully Qualified Domain Name (FQDN). It is implemented by the following class: `synapse.models.inet.Fqdn`.

An example of `inet:fqdn`:

- `vertex.link`

inet:http:cookie

An individual HTTP cookie string. It is implemented by the following class: `synapse.models.inet.HttpCookie`.

An example of `inet:http:cookie`:

- `PHPSESSID=el4ukv0kqbvoirg7nkp4dncpk3`

inet:ipv4

An IPv4 address. It is implemented by the following class: `synapse.models.inet.IPv4`.

An example of `inet:ipv4`:

- `1.2.3.4`

inet:ipv4range

An IPv4 address range. It is implemented by the following class: `synapse.models.inet.IPv4Range`.

An example of `inet:ipv4range`:

- `1.2.3.4-1.2.3.8`

inet:ipv6

An IPv6 address. It is implemented by the following class: `synapse.models.inet.IPv6`.

An example of `inet:ipv6`:

- `2607:f8b0:4004:809::200e`

inet:ipv6range

An IPv6 address range. It is implemented by the following class: `synapse.models.inet.IPv6Range`.

An example of `inet:ipv6range`:

- `(2607:f8b0:4004:809::200e, 2607:f8b0:4004:809::2011)`

inet:rfc2822:addr

An RFC 2822 Address field. It is implemented by the following class: `synapse.models.inet.Rfc2822Addr`.

An example of `inet:rfc2822:addr`:

- "Visi Kenshoto" <visi@vertex.link>

inet:url

A Universal Resource Locator (URL). It is implemented by the following class: `synapse.models.inet.Url`.

An example of `inet:url`:

- `http://www.woot.com/files/index.html`

int

The base 64 bit signed integer type. It is implemented by the following class: `synapse.lib.types.Int`.

The base type `int` has the following default options set:

- size: `8`
- signed: `True`
- enums:strict: `True`
- fmt: `%d`
- min: `None`
- max: `None`
- ismin: `False`
- ismax: `False`

it:sec:cpe

A NIST CPE 2.3 Formatted String. It is implemented by the following class: `synapse.models.infotech.Cpe23Str`.

The base type `it:sec:cpe` has the following default options set:

- lower: `True`

it:sec:cpe:v2_2

A NIST CPE 2.2 Formatted String. It is implemented by the following class: `synapse.models.infotech.Cpe22Str`.

The base type `it:sec:cpe:v2_2` has the following default options set:

- lower: `True`

it:semver

Semantic Version type. It is implemented by the following class: `synapse.models.infotech.SemVer`.

ival

A time window/interval. It is implemented by the following class: `synapse.lib.types.Ival`.

loc

The base geo political location type. It is implemented by the following class: `synapse.lib.types.Loc`.

ndef

The node definition type for a (form,valu) compound field. It is implemented by the following class: `synapse.lib.types.Ndef`.

nodeprop

The nodeprop type for a (prop,valu) compound field. It is implemented by the following class: `synapse.lib.types.NodeProp`.

range

A base range type. It is implemented by the following class: `synapse.lib.types.Range`.

The base type `range` has the following default options set:

- type: `('int', {})`

str

The base string type. It is implemented by the following class: `synapse.lib.types.Str`.

The base type `str` has the following default options set:

- enums: `None`
- regex: `None`
- lower: `False`
- strip: `False`
- replace: `()`

- onespace: False
- globsuffix: False

syn:tag

The base type for a synapse tag. It is implemented by the following class: `synapse.lib.types.Tag`.

The base type `syn:tag` has the following default options set:

- enums: None
- regex: None
- lower: False
- strip: False
- replace: ()
- onespace: False
- globsuffix: False

syn:tag:part

A tag component string. It is implemented by the following class: `synapse.lib.types.TagPart`.

The base type `syn:tag:part` has the following default options set:

- enums: None
- regex: None
- lower: False
- strip: False
- replace: ()
- onespace: False
- globsuffix: False

taxon

A component of a hierarchical taxonomy. It is implemented by the following class: `synapse.lib.types.Taxon`.

The base type `taxon` has the following default options set:

- enums: None
- regex: None
- lower: False
- strip: False
- replace: ()
- onespace: False
- globsuffix: False

taxonomy

A hierarchical taxonomy. It is implemented by the following class: `synapse.lib.types.Taxonomy`.

The base type `taxonomy` has the following default options set:

- enums: `None`
- regex: `None`
- lower: `False`
- strip: `False`
- replace: `()`
- onespace: `False`
- globsuffix: `False`

tel:mob:imei

An International Mobile Equipment Id. It is implemented by the following class: `synapse.models.telco.Imei`.

An example of `tel:mob:imei`:

- `490154203237518`

tel:mob:imsi

An International Mobile Subscriber Id. It is implemented by the following class: `synapse.models.telco.Imsi`.

An example of `tel:mob:imsi`:

- `310150123456789`

tel:phone

A phone number. It is implemented by the following class: `synapse.models.telco.Phone`.

An example of `tel:phone`:

- `+15558675309`

time

A date/time value. It is implemented by the following class: `synapse.lib.types.Time`.

The base type `time` has the following default options set:

- ismin: `False`
- ismax: `False`

timeedge

An digraph edge base type with a unique time. It is implemented by the following class: `synapse.lib.types.TimeEdge`.

velocity

A velocity with base units in mm/sec. It is implemented by the following class: `synapse.lib.types.Velocity`.

The base type `velocity` has the following default options set:

- relative: `False`

12.1.2 Types

Regular types are derived from BaseTypes.

auth:access

An instance of using creds to access a resource. The `auth:access` type is derived from the base type: `guid`.

auth:creds

A unique set of credentials used to access a resource. The `auth:creds` type is derived from the base type: `guid`.

belief:subscriber

A contact which subscribes to a belief system. The `belief:subscriber` type is derived from the base type: `guid`.

belief:system

A belief system such as an ideology, philosophy, or religion. The `belief:system` type is derived from the base type: `guid`.

belief:system:type:taxonomy

A hierarchical taxonomy of belief system types. The `belief:system:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `belief:system:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

belief:tenet

A concrete tenet potentially shared by multiple belief systems. The `belief:tenet` type is derived from the base type: `guid`.

biz:bundle

A bundle allows construction of products which bundle instances of other products. The `biz:bundle` type is derived from the base type: `guid`.

biz:deal

A sales or procurement effort in pursuit of a purchase. The `biz:deal` type is derived from the base type: `guid`.

biz:dealstatus

A deal/rfp status taxonomy. The `biz:dealstatus` type is derived from the base type: `taxonomy`.

The type `biz:dealstatus` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

biz:dealtype

A deal type taxonomy. The `biz:dealtype` type is derived from the base type: `taxonomy`.

The type `biz:dealtype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

biz:listing

A product or service being listed for sale at a given price by a specific seller. The `biz:listing` type is derived from the base type: `guid`.

biz:prodtype

A product type taxonomy. The `biz:prodtype` type is derived from the base type: `taxonomy`.

The type `biz:prodtype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

biz:product

A product which is available for purchase. The `biz:product` type is derived from the base type: `guid`.

biz:rfp

An RFP (Request for Proposal) soliciting proposals. The `biz:rfp` type is derived from the base type: `guid`.

biz:service

A service which is performed by a specific organization. The `biz:service` type is derived from the base type: `guid`.

biz:service:type:taxonomy

A taxonomy of service offering types. The `biz:service:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `biz:service:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

biz:stake

A stake or partial ownership in a company. The `biz:stake` type is derived from the base type: `guid`.

crypto:algorithm

A cryptographic algorithm name. The `crypto:algorithm` type is derived from the base type: `str`.

An example of `crypto:algorithm`:

- `aes256`

The type `crypto:algorithm` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

crypto:currency:address

An individual crypto currency address. The `crypto:currency:address` type is derived from the base type: `comp`.

An example of `crypto:currency:address`:

- `btc/1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2`

The type `crypto:currency:address` has the following options set:

- fields: `(('coin', 'crypto:currency:coin'), ('iden', 'str'))`
- sepr: `/`

crypto:currency:block

An individual crypto currency block record on the blockchain. The `crypto:currency:block` type is derived from the base type: `comp`.

The type `crypto:currency:block` has the following options set:

- fields: `(('coin', 'crypto:currency:coin'), ('offset', 'int'))`
- sepr: `/`

crypto:currency:client

A fused node representing a crypto currency address used by an Internet client. The `crypto:currency:client` type is derived from the base type: `comp`.

An example of `crypto:currency:client`:

- `(1.2.3.4, (btc, 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2))`

The type `crypto:currency:client` has the following options set:

- fields: `(('inetaddr', 'inet:client'), ('coinaddr', 'crypto:currency:address'))`

crypto:currency:coin

An individual crypto currency type. The `crypto:currency:coin` type is derived from the base type: `str`.

An example of `crypto:currency:coin`:

- `btc`

The type `crypto:currency:coin` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

crypto:currency:transaction

An individual crypto currency transaction recorded on the blockchain. The `crypto:currency:transaction` type is derived from the base type: `guid`.

crypto:key

A cryptographic key and algorithm. The `crypto:key` type is derived from the base type: `guid`.

crypto:payment:input

A payment made into a transaction. The `crypto:payment:input` type is derived from the base type: `guid`.

crypto:payment:output

A payment received from a transaction. The `crypto:payment:output` type is derived from the base type: `guid`.

crypto:smart:contract

A smart contract. The `crypto:smart:contract` type is derived from the base type: `guid`.

crypto:smart:effect:burntoken

A smart contract effect which destroys a non-fungible token. The `crypto:smart:effect:burntoken` type is derived from the base type: `guid`.

crypto:smart:effect:edittokensupply

A smart contract effect which increases or decreases the supply of a fungible token. The `crypto:smart:effect:edittokensupply` type is derived from the base type: `guid`.

crypto:smart:effect:minttoken

A smart contract effect which creates a new non-fungible token. The `crypto:smart:effect:minttoken` type is derived from the base type: `guid`.

crypto:smart:effect:proxytoken

A smart contract effect which grants a non-owner address the ability to manipulate a specific non-fungible token. The `crypto:smart:effect:proxytoken` type is derived from the base type: `guid`.

crypto:smart:effect:proxytokenall

A smart contract effect which grants a non-owner address the ability to manipulate all non-fungible tokens of the owner. The `crypto:smart:effect:proxytokenall` type is derived from the base type: `guid`.

crypto:smart:effect:proxytokens

A smart contract effect which grants a non-owner address the ability to manipulate fungible tokens. The `crypto:smart:effect:proxytokens` type is derived from the base type: `guid`.

crypto:smart:effect:transfertoken

A smart contract effect which transfers ownership of a non-fungible token. The `crypto:smart:effect:transfertoken` type is derived from the base type: `guid`.

crypto:smart:effect:transfertokens

A smart contract effect which transfers fungible tokens. The `crypto:smart:effect:transfertokens` type is derived from the base type: `guid`.

crypto:smart:token

A token managed by a smart contract. The `crypto:smart:token` type is derived from the base type: `comp`.

The type `crypto:smart:token` has the following options set:

- fields: `(('contract', 'crypto:smart:contract'), ('tokenid', 'hugenum'))`

crypto:x509:cert

A unique X.509 certificate. The `crypto:x509:cert` type is derived from the base type: `guid`.

crypto:x509:crl

A unique X.509 Certificate Revocation List. The `crypto:x509:crl` type is derived from the base type: `guid`.

crypto:x509:revoked

A revocation relationship between a CRL and an X.509 certificate. The `crypto:x509:revoked` type is derived from the base type: `comp`.

The type `crypto:x509:revoked` has the following options set:

- fields: `(('crl', 'crypto:x509:crl'), ('cert', 'crypto:x509:cert'))`

crypto:x509:san

An X.509 Subject Alternative Name (SAN). The `crypto:x509:san` type is derived from the base type: `comp`.

The type `crypto:x509:san` has the following options set:

- fields: `(('type', 'str'), ('value', 'str'))`

crypto:x509:signedfile

A digital signature relationship between an X.509 certificate and a file. The `crypto:x509:signedfile` type is derived from the base type: `comp`.

The type `crypto:x509:signedfile` has the following options set:

- fields: `(('cert', 'crypto:x509:cert'), ('file', 'file:bytes'))`

econ:acct:balance

A snapshot of the balance of an account at a point in time. The `econ:acct:balance` type is derived from the base type: `guid`.

econ:acct:invoice

An invoice issued requesting payment. The `econ:acct:invoice` type is derived from the base type: `guid`.

econ:acct:payment

A payment or crypto currency transaction. The `econ:acct:payment` type is derived from the base type: `guid`.

econ:acct:receipt

A receipt issued as proof of payment. The `econ:acct:receipt` type is derived from the base type: `guid`.

econ:acquired

Deprecated. Please use econ:purchase -(acquired)> *. The `econ:acquired` type is derived from the base type: `comp`.

The type `econ:acquired` has the following options set:

- fields: `(('purchase', 'econ:purchase'), ('item', 'ndef'))`

econ:bank:aba:rtn

An American Bank Association (ABA) routing transit number (RTN). The `econ:bank:aba:rtn` type is derived from the base type: `str`.

The type `econ:bank:aba:rtn` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `[0-9]{9}`
- replace: `()`
- strip: `False`

econ:bank:account

A bank account. The `econ:bank:account` type is derived from the base type: `guid`.

econ:bank:account:type:taxonomy

A bank account type taxonomy. The `econ:bank:account:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `econ:bank:account:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

econ:bank:balance

A balance contained by a bank account at a point in time. The `econ:bank:balance` type is derived from the base type: `guid`.

econ:bank:iban

An International Bank Account Number. The `econ:bank:iban` type is derived from the base type: `str`.

The type `econ:bank:iban` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `[A-Z]{2}[0-9]{2}[a-zA-Z0-9]{1,30}`
- replace: `()`
- strip: `False`

econ:bank:statement

A statement of bank account payment activity over a period of time. The `econ:bank:statement` type is derived from the base type: `guid`.

econ:bank:swift:bic

A Society for Worldwide Interbank Financial Telecommunication (SWIFT) Business Identifier Code (BIC). The `econ:bank:swift:bic` type is derived from the base type: `str`.

The type `econ:bank:swift:bic` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `[A-Z]{6}[A-Z0-9]{5}`

- replace: `()`
- strip: `False`

econ:currency

The name of a system of money in general use. The `econ:currency` type is derived from the base type: `str`.

An example of `econ:currency`:

- `usd`

The type `econ:currency` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

econ:fin:bar

A sample of the open, close, high, low prices of a security in a specific time window. The `econ:fin:bar` type is derived from the base type: `guid`.

econ:fin:exchange

A financial exchange where securities are traded. The `econ:fin:exchange` type is derived from the base type: `guid`.

econ:fin:security

A financial security which is typically traded on an exchange. The `econ:fin:security` type is derived from the base type: `guid`.

econ:fin:tick

A sample of the price of a security at a single moment in time. The `econ:fin:tick` type is derived from the base type: `guid`.

econ:pay:card

A single payment card. The `econ:pay:card` type is derived from the base type: `guid`.

econ:pay:cvv

A Card Verification Value (CVV). The `econ:pay:cvv` type is derived from the base type: `str`.

The type `econ:pay:cvv` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[0-9]{1,6}$`
- replace: `()`
- strip: `False`

econ:pay:iin

An Issuer Id Number (IIN). The `econ:pay:iin` type is derived from the base type: `int`.

The type `econ:pay:iin` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `999999`
- min: `0`
- signed: `True`
- size: `8`

econ:pay:mii

A Major Industry Identifier (MII). The `econ:pay:mii` type is derived from the base type: `int`.

The type `econ:pay:mii` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `9`
- min: `0`
- signed: `True`
- size: `8`

econ:pay:pan

A Primary Account Number (PAN) or card number. The `econ:pay:pan` type is derived from the base type: `str`.

The type `econ:pay:pan` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^(?<iin>(?<mii>[0-9]{1})[0-9]{5})[0-9]{1,13}$`
- replace: `()`
- strip: `False`

econ:pay:pin

A Personal Identification Number (PIN). The `econ:pay:pin` type is derived from the base type: `str`.

The type `econ:pay:pin` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[0-9]{3,6}$`
- replace: `()`
- strip: `False`

econ:price

The amount of money expected, required, or given in payment for something. The `econ:price` type is derived from the base type: `hugenum`.

An example of `econ:price`:

- `2.20`

The type `econ:price` has the following options set:

- modulo: `None`
- norm: `False`
- units: `None`

econ:purchase

A purchase event. The `econ:purchase` type is derived from the base type: `guid`.

econ:receipt:item

A line item included as part of a purchase. The `econ:receipt:item` type is derived from the base type: `guid`.

edge:has

A digraph edge which records that N1 has N2. The `edge:has` type is derived from the base type: `edge`.

edge:refs

A digraph edge which records that N1 refers to or contains N2. The `edge:refs` type is derived from the base type: `edge`.

edge:wentto

A digraph edge which records that N1 went to N2 at a specific time. The `edge:wentto` type is derived from the base type: `timeedge`.

edu:class

An instance of an edu:course taught at a given time. The `edu:class` type is derived from the base type: `guid`.

edu:course

A course of study taught by an org. The `edu:course` type is derived from the base type: `guid`.

file:archive:entry

An archive entry representing a file and metadata within a parent archive file. The `file:archive:entry` type is derived from the base type: `guid`.

file:filepath

The fused knowledge of the association of a file:bytes node and a file:path. The `file:filepath` type is derived from the base type: `comp`.

The type `file:filepath` has the following options set:

- fields: `(('file', 'file:bytes'), ('path', 'file:path'))`

file:ismime

Records one, of potentially multiple, mime types for a given file. The `file:ismime` type is derived from the base type: `comp`.

The type `file:ismime` has the following options set:

- fields: `(('file', 'file:bytes'), ('mime', 'file:mime'))`

file:mime

A file mime name string. The `file:mime` type is derived from the base type: `str`.

An example of `file:mime`:

- `text/plain`

The type `file:mime` has the following options set:

- globsuffix: `False`
- lower: `1`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

file:mime:gif

The GUID of a set of mime metadata for a .gif file. The `file:mime:gif` type is derived from the base type: `guid`.

file:mime:jpg

The GUID of a set of mime metadata for a .jpg file. The `file:mime:jpg` type is derived from the base type: `guid`.

file:mime:macho:loadcmd

A generic load command pulled from the Mach-O headers. The `file:mime:macho:loadcmd` type is derived from the base type: `guid`.

file:mime:macho:section

A section inside a Mach-O binary denoting a named region of bytes inside a segment. The `file:mime:macho:section` type is derived from the base type: `guid`.

file:mime:macho:segment

A named region of bytes inside a Mach-O binary. The `file:mime:macho:segment` type is derived from the base type: `guid`.

file:mime:macho:uuid

A specific load command denoting a UUID used to uniquely identify the Mach-O binary. The `file:mime:macho:uuid` type is derived from the base type: `guid`.

file:mime:macho:version

A specific load command used to denote the version of the source used to build the Mach-O binary. The `file:mime:macho:version` type is derived from the base type: `guid`.

file:mime:msdoc

The GUID of a set of mime metadata for a Microsoft Word file. The `file:mime:msdoc` type is derived from the base type: `guid`.

file:mime:msppt

The GUID of a set of mime metadata for a Microsoft Powerpoint file. The `file:mime:msppt` type is derived from the base type: `guid`.

file:mime:msxls

The GUID of a set of mime metadata for a Microsoft Excel file. The `file:mime:msxls` type is derived from the base type: `guid`.

file:mime:pe:export

The fused knowledge of a file:bytes node containing a pe named export. The `file:mime:pe:export` type is derived from the base type: `comp`.

The type `file:mime:pe:export` has the following options set:

- fields: `(('file', 'file:bytes'), ('name', 'str'))`

file:mime:pe:resource

The fused knowledge of a file:bytes node containing a pe resource. The `file:mime:pe:resource` type is derived from the base type: `comp`.

The type `file:mime:pe:resource` has the following options set:

  - fields:

```
[
 [
  "file",
  "file:bytes"
 ],
 [
  "type",
  "pe:resource:type"
 ],
 [
  "langid",
  "pe:langid"
 ],
 [
  "resource",
  "file:bytes"
 ]
]
```


file:mime:pe:section

The fused knowledge a file:bytes node containing a pe section. The `file:mime:pe:section` type is derived from the base type: `comp`.

The type `file:mime:pe:section` has the following options set:

- fields: `(('file', 'file:bytes'), ('name', 'str'), ('sha256', 'hash:sha256'))`

file:mime:pe:vsvers:info

knowledge of a file:bytes node containing vsvers info. The `file:mime:pe:vsvers:info` type is derived from the base type: `comp`.

The type `file:mime:pe:vsvers:info` has the following options set:

- fields: `(('file', 'file:bytes'), ('keyval', 'file:mime:pe:vsvers:keyval'))`

file:mime:pe:vsvers:keyval

A key value pair found in a PE vsversion info structure. The `file:mime:pe:vsvers:keyval` type is derived from the base type: `comp`.

The type `file:mime:pe:vsvers:keyval` has the following options set:

- fields: `(('name', 'str'), ('value', 'str'))`

file:mime:png

The GUID of a set of mime metadata for a .png file. The `file:mime:png` type is derived from the base type: `guid`.

file:mime:rtf

The GUID of a set of mime metadata for a .rtf file. The `file:mime:rtf` type is derived from the base type: `guid`.

file:mime:tif

The GUID of a set of mime metadata for a .tif file. The `file:mime:tif` type is derived from the base type: `guid`.

file:string

Deprecated. Please use the edge -(refs)> it:dev:str. The `file:string` type is derived from the base type: `comp`.

The type `file:string` has the following options set:

- fields: `(('file', 'file:bytes'), ('string', 'str'))`

file:subfile

A parent file that fully contains the specified child file. The `file:subfile` type is derived from the base type: `comp`.

The type `file:subfile` has the following options set:

- fields: `(('parent', 'file:bytes'), ('child', 'file:bytes'))`

geo:address

A street/mailing address string. The `geo:address` type is derived from the base type: `str`.

The type `geo:address` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

geo:altitude

A negative or positive offset from Mean Sea Level (6,371.0088km from Earths core). The `geo:altitude` type is derived from the base type: `geo:dist`.

An example of `geo:altitude`:

- `10 km`

The type `geo:altitude` has the following options set:

- baseoff: `6371008800`
- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

geo:bbox

A geospatial bounding box in (xmin, xmax, ymin, ymax) format. The `geo:bbox` type is derived from the base type: `comp`.

The type `geo:bbox` has the following options set:

- fields:

```
[
 [
  "xmin",
  "geo:longitude"
 ],
 [
  "xmax",
  "geo:longitude"
 ],
 [
  "ymin",
  "geo:latitude"
 ],
 [
  "ymax",
  "geo:latitude"
 ]
]
```

- sepr: `,`

geo:json

GeoJSON structured JSON data. The `geo:json` type is derived from the base type: `data`.

The type `geo:json` has the following options set:

- schema:

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "definitions": {
  "BoundingBox": {
   "items": {
    "type": "number"
   },
   "minItems": 4,
   "type": "array"
  },
  "Feature": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
    },
    "geometry": {
     "oneOf": [
      {
       "type": "null"
      },
      {
       "$ref": "#/definitions/Point"
      },
      {
       "$ref": "#/definitions/LineString"
      },
      {
       "$ref": "#/definitions/Polygon"
      },
      {
       "$ref": "#/definitions/MultiPoint"
      },
      {
       "$ref": "#/definitions/MultiLineString"
      },
      {
       "$ref": "#/definitions/MultiPolygon"
      },
      {
       "$ref": "#/definitions/GeometryCollection"
      }
     ]
    },
    "properties": {
     "oneOf": [
      {
```

(continues on next page)

(continued from previous page)

```
     "type": "null"
    },
    {
     "type": "object"
    }
   ]
  },
  "type": {
   "enum": [
    "Feature"
   ],
   "type": "string"
  }
 },
 "required": [
  "type",
  "properties",
  "geometry"
 ],
 "title": "GeoJSON Feature",
 "type": "object"
},
"FeatureCollection": {
 "properties": {
  "bbox": {
   "$ref": "#/definitions/BoundingBox"
  },
  "features": {
   "items": {
    "$ref": "#/definitions/Feature"
   },
   "type": "array"
  },
  "type": {
   "enum": [
    "FeatureCollection"
   ],
   "type": "string"
  }
 },
 "required": [
  "type",
  "features"
 ],
 "title": "GeoJSON FeatureCollection",
 "type": "object"
},
"GeometryCollection": {
 "properties": {
  "bbox": {
   "$ref": "#/definitions/BoundingBox"
  },
```

(continues on next page)

(continued from previous page)


```
    "geometries": {
     "items": {
      "oneOf": [
       {
        "$ref": "#/definitions/Point"
       },
       {
        "$ref": "#/definitions/LineString"
       },
       {
        "$ref": "#/definitions/Polygon"
       },
       {
        "$ref": "#/definitions/MultiPoint"
       },
       {
        "$ref": "#/definitions/MultiLineString"
       },
       {
        "$ref": "#/definitions/MultiPolygon"
       }
      ]
     },
     "type": "array"
    },
    "type": {
     "enum": [
      "GeometryCollection"
     ],
     "type": "string"
    }
   },
   "required": [
    "type",
    "geometries"
   ],
   "title": "GeoJSON GeometryCollection",
   "type": "object"
  },
  "LineString": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
    },
    "coordinates": {
     "$ref": "#/definitions/LineStringCoordinates"
    },
    "type": {
     "enum": [
      "LineString"
     ],
     "type": "string"
```


(continues on next page)

(continued from previous page)


```
    }
   },
   "required": [
    "type",
    "coordinates"
   ],
   "title": "GeoJSON LineString",
   "type": "object"
  },
  "LineStringCoordinates": {
   "items": {
    "$ref": "#/definitions/PointCoordinates"
   },
   "minItems": 2,
   "type": "array"
  },
  "LinearRingCoordinates": {
   "items": {
    "$ref": "#/definitions/PointCoordinates"
   },
   "minItems": 4,
   "type": "array"
  },
  "MultiLineString": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
    },
    "coordinates": {
     "items": {
      "$ref": "#/definitions/LineStringCoordinates"
     },
     "type": "array"
    },
    "type": {
     "enum": [
      "MultiLineString"
     ],
     "type": "string"
    }
   },
   "required": [
    "type",
    "coordinates"
   ],
   "title": "GeoJSON MultiLineString",
   "type": "object"
  },
  "MultiPoint": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
```


(continues on next page)

(continued from previous page)

```
    },
    "coordinates": {
     "items": {
      "$ref": "#/definitions/PointCoordinates"
     },
     "type": "array"
    },
    "type": {
     "enum": [
      "MultiPoint"
     ],
     "type": "string"
    }
   },
   "required": [
    "type",
    "coordinates"
   ],
   "title": "GeoJSON MultiPoint",
   "type": "object"
  },
  "MultiPolygon": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
    },
    "coordinates": {
     "items": {
      "$ref": "#/definitions/PolygonCoordinates"
     },
     "type": "array"
    },
    "type": {
     "enum": [
      "MultiPolygon"
     ],
     "type": "string"
    }
   },
   "required": [
    "type",
    "coordinates"
   ],
   "title": "GeoJSON MultiPolygon",
   "type": "object"
  },
  "Point": {
   "properties": {
    "bbox": {
     "$ref": "#/definitions/BoundingBox"
    },
    "coordinates": {
```

(continues on next page)

(continued from previous page)

```
      "$ref": "#/definitions/PointCoordinates"
    },
    "type": {
      "enum": [
        "Point"
      ],
      "type": "string"
    }
  },
  "required": [
    "type",
    "coordinates"
  ],
  "title": "GeoJSON Point",
  "type": "object"
},
"PointCoordinates": {
  "items": {
    "type": "number"
  },
  "minItems": 2,
  "type": "array"
},
"Polygon": {
  "properties": {
    "bbox": {
      "$ref": "#/definitions/BoundingBox"
    },
    "coordinates": {
      "$ref": "#/definitions/PolygonCoordinates"
    },
    "type": {
      "enum": [
        "Polygon"
      ],
      "type": "string"
    }
  },
  "required": [
    "type",
    "coordinates"
  ],
  "title": "GeoJSON Polygon",
  "type": "object"
},
"PolygonCoordinates": {
  "items": {
    "$ref": "#/definitions/LinearRingCoordinates"
  },
  "type": "array"
}
},
```

(continues on next page)

(continued from previous page)


```
 "oneOf": [
  {
   "$ref": "#/definitions/Point"
  },
  {
   "$ref": "#/definitions/LineString"
  },
  {
   "$ref": "#/definitions/Polygon"
  },
  {
   "$ref": "#/definitions/MultiPoint"
  },
  {
   "$ref": "#/definitions/MultiLineString"
  },
  {
   "$ref": "#/definitions/MultiPolygon"
  },
  {
   "$ref": "#/definitions/GeometryCollection"
  },
  {
   "$ref": "#/definitions/Feature"
  },
  {
   "$ref": "#/definitions/FeatureCollection"
  }
 ]
}
```


geo:latitude

A latitude in floating point notation. The `geo:latitude` type is derived from the base type: `float`.

An example of `geo:latitude`:

- `31.337`

The type `geo:latitude` has the following options set:

- fmt: `%f`
- max: `90.0`
- maxisvalid: `True`
- min: `-90.0`
- minisvalid: `True`

geo:longitude

A longitude in floating point notation. The `geo:longitude` type is derived from the base type: `float`.

An example of `geo:longitude`:

- `31.337`

The type `geo:longitude` has the following options set:

- fmt: `%f`
- max: `180.0`
- maxisvalid: `True`
- min: `-180.0`
- minisvalid: `False`

geo:name

An unstructured place name or address. The `geo:name` type is derived from the base type: `str`.

The type `geo:name` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

geo:nloc

Records a node latitude/longitude in space-time. The `geo:nloc` type is derived from the base type: `comp`.

The type `geo:nloc` has the following options set:

- fields: `(('ndef', 'ndef'), ('latlong', 'geo:latlong'), ('time', 'time'))`

geo:place

A GUID for a geographic place. The `geo:place` type is derived from the base type: `guid`.

geo:place:taxonomy

A taxonomy of place types. The `geo:place:taxonomy` type is derived from the base type: `taxonomy`.

The type `geo:place:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

geo:telem

A geospatial position of a node at a given time. The node should be linked via -(seenat)> edges. The `geo:telem` type is derived from the base type: `guid`.

gov:cn:icp

A Chinese Internet Content Provider ID. The `gov:cn:icp` type is derived from the base type: `int`.

The type `gov:cn:icp` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

gov:cn:mucd

A Chinese PLA MUCD. The `gov:cn:mucd` type is derived from the base type: `int`.

The type `gov:cn:mucd` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`

- size: 8

gov:intl:un:m49

UN M49 Numeric Country Code. The `gov:intl:un:m49` type is derived from the base type: `int`.

The type `gov:intl:un:m49` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `999`
- min: `1`
- signed: `True`
- size: `8`

gov:us:cage

A Commercial and Government Entity (CAGE) code. The `gov:us:cage` type is derived from the base type: `str`.

The type `gov:us:cage` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

gov:us:ssn

A US Social Security Number (SSN). The `gov:us:ssn` type is derived from the base type: `int`.

The type `gov:us:ssn` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

gov:us:zip

A US Postal Zip Code. The `gov:us:zip` type is derived from the base type: `int`.

The type `gov:us:zip` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

graph:cluster

A generic node, used in conjunction with Edge types, to cluster arbitrary nodes to a single node in the model. The `graph:cluster` type is derived from the base type: `guid`.

graph:edge

A generic digraph edge to show relationships outside the model. The `graph:edge` type is derived from the base type: `edge`.

graph:event

A generic event node to represent events outside the model. The `graph:event` type is derived from the base type: `guid`.

graph:node

A generic node used to represent objects outside the model. The `graph:node` type is derived from the base type: `guid`.

graph:timeedge

A generic digraph time edge to show relationships outside the model. The `graph:timeedge` type is derived from the base type: `timeedge`.

hash:lm

A hex encoded Microsoft Windows LM password hash. The `hash:lm` type is derived from the base type: `hex`.

An example of `hash:lm`:

- `d41d8cd98f00b204e9800998ecf8427e`

The type `hash:lm` has the following options set:

- size: `32`
- zeropad: `0`

hash:md5

A hex encoded MD5 hash. The `hash:md5` type is derived from the base type: `hex`.

An example of `hash:md5`:

- `d41d8cd98f00b204e9800998ecf8427e`

The type `hash:md5` has the following options set:

- size: `32`
- zeropad: `0`

hash:ntlm

A hex encoded Microsoft Windows NTLM password hash. The `hash:ntlm` type is derived from the base type: `hex`.

An example of `hash:ntlm`:

- `d41d8cd98f00b204e9800998ecf8427e`

The type `hash:ntlm` has the following options set:

- size: `32`
- zeropad: `0`

hash:sha1

A hex encoded SHA1 hash. The `hash:sha1` type is derived from the base type: `hex`.

An example of `hash:sha1`:

- `da39a3ee5e6b4b0d3255bfef95601890afd80709`

The type `hash:sha1` has the following options set:

- size: `40`
- zeropad: `0`

hash:sha256

A hex encoded SHA256 hash. The `hash:sha256` type is derived from the base type: `hex`.

An example of `hash:sha256`:

- `ad9f4fe922b61e674a09530831759843b1880381de686a43460a76864ca0340c`

The type `hash:sha256` has the following options set:

- size: `64`
- zeropad: `0`

hash:sha384

A hex encoded SHA384 hash. The `hash:sha384` type is derived from the base type: `hex`.

An example of `hash:sha384`:

- `d425f1394e418ce01ed1579069a8bfaa1da8f32cf823982113ccbef531fa36bda9987f389c5af05b5e28035242efab6c`

The type `hash:sha384` has the following options set:

- size: `96`
- zeropad: `0`

hash:sha512

A hex encoded SHA512 hash. The `hash:sha512` type is derived from the base type: `hex`.

An example of `hash:sha512`:

- `ca74fe2ff2d03b29339ad7d08ba21d192077fece1715291c7b43c20c9136cd132788239189f3441a87eb23ce2660aa243f3`

The type `hash:sha512` has the following options set:

- size: `128`
- zeropad: `0`

inet:asn

An Autonomous System Number (ASN). The `inet:asn` type is derived from the base type: `int`.

The type `inet:asn` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

inet:asnet4

An Autonomous System Number (ASN) and its associated IPv4 address range. The `inet:asnet4` type is derived from the base type: `comp`.

An example of `inet:asnet4`:

- `(54959, (1.2.3.4, 1.2.3.20))`

The type `inet:asnet4` has the following options set:

- fields: `(('asn', 'inet:asn'), ('net4', 'inet:net4'))`

inet:asnet6

An Autonomous System Number (ASN) and its associated IPv6 address range. The `inet:asnet6` type is derived from the base type: `comp`.

An example of `inet:asnet6`:

- `(54959, (ff::00, ff::02))`

The type `inet:asnet6` has the following options set:

- fields: `(('asn', 'inet:asn'), ('net6', 'inet:net6'))`

inet:banner

A network protocol banner string presented by a server. The `inet:banner` type is derived from the base type: `comp`.

The type `inet:banner` has the following options set:

- fields: `(('server', 'inet:server'), ('text', 'it:dev:str'))`

inet:client

A network client address. The `inet:client` type is derived from the base type: `inet:addr`.

An example of `inet:client`:

- `tcp://1.2.3.4:80`

The type `inet:client` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:dns:a

The result of a DNS A record lookup. The `inet:dns:a` type is derived from the base type: `comp`.

An example of `inet:dns:a`:

- `(vertex.link,1.2.3.4)`

The type `inet:dns:a` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('ipv4', 'inet:ipv4'))`

inet:dns:aaaa

The result of a DNS AAAA record lookup. The `inet:dns:aaaa` type is derived from the base type: `comp`.

An example of `inet:dns:aaaa`:

- `(vertex.link,2607:f8b0:4004:809::200e)`

The type `inet:dns:aaaa` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('ipv6', 'inet:ipv6'))`

inet:dns:answer

A single answer from within a DNS reply. The `inet:dns:answer` type is derived from the base type: `guid`.

inet:dns:cname

The result of a DNS CNAME record lookup. The `inet:dns:cname` type is derived from the base type: `comp`.

An example of `inet:dns:cname`:

- `(foo.vertex.link,vertex.link)`

The type `inet:dns:cname` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('cname', 'inet:fqdn'))`

inet:dns:dynreg

A dynamic DNS registration. The `inet:dns:dynreg` type is derived from the base type: `guid`.

inet:dns:mx

The result of a DNS MX record lookup. The `inet:dns:mx` type is derived from the base type: `comp`.

An example of `inet:dns:mx`:

- `(vertex.link,mail.vertex.link)`

The type `inet:dns:mx` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('mx', 'inet:fqdn'))`

inet:dns:ns

The result of a DNS NS record lookup. The `inet:dns:ns` type is derived from the base type: `comp`.

An example of `inet:dns:ns`:

- `(vertex.link,ns.dnshost.com)`

The type `inet:dns:ns` has the following options set:

- fields: `(('zone', 'inet:fqdn'), ('ns', 'inet:fqdn'))`

inet:dns:query

A DNS query unique to a given client. The `inet:dns:query` type is derived from the base type: `comp`.

An example of `inet:dns:query`:

- `(1.2.3.4, woot.com, 1)`

The type `inet:dns:query` has the following options set:

- fields: `(('client', 'inet:client'), ('name', 'inet:dns:name'), ('type', 'int'))`

inet:dns:request

A single instance of a DNS resolver request and optional reply info. The `inet:dns:request` type is derived from the base type: `guid`.

inet:dns:rev

The transformed result of a DNS PTR record lookup. The `inet:dns:rev` type is derived from the base type: `comp`.

An example of `inet:dns:rev`:

- `(1.2.3.4,vertex.link)`

The type `inet:dns:rev` has the following options set:

- fields: `(('ipv4', 'inet:ipv4'), ('fqdn', 'inet:fqdn'))`

inet:dns:rev6

The transformed result of a DNS PTR record for an IPv6 address. The `inet:dns:rev6` type is derived from the base type: `comp`.

An example of `inet:dns:rev6`:

- `(2607:f8b0:4004:809::200e,vertex.link)`

The type `inet:dns:rev6` has the following options set:

- fields: `(('ipv6', 'inet:ipv6'), ('fqdn', 'inet:fqdn'))`

inet:dns:soa

The result of a DNS SOA record lookup. The `inet:dns:soa` type is derived from the base type: `guid`.

inet:dns:txt

The result of a DNS MX record lookup. The `inet:dns:txt` type is derived from the base type: `comp`.

An example of `inet:dns:txt`:

- `(hehe.vertex.link,"fancy TXT record")`

The type `inet:dns:txt` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('txt', 'str'))`

inet:dns:type

A DNS query/answer type integer. The `inet:dns:type` type is derived from the base type: `int`.

The type `inet:dns:type` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

inet:dns:wild:a

A DNS A wild card record and the IPv4 it resolves to. The `inet:dns:wild:a` type is derived from the base type: `comp`.

The type `inet:dns:wild:a` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('ipv4', 'inet:ipv4'))`

inet:dns:wild:aaaa

A DNS AAAA wild card record and the IPv6 it resolves to. The `inet:dns:wild:aaaa` type is derived from the base type: `comp`.

The type `inet:dns:wild:aaaa` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('ipv6', 'inet:ipv6'))`

inet:download

An instance of a file downloaded from a server. The `inet:download` type is derived from the base type: `guid`.

inet:egress

A host using a specific network egress client address. The `inet:egress` type is derived from the base type: `guid`.

inet:email:header

A unique email message header. The `inet:email:header` type is derived from the base type: `comp`.

The type `inet:email:header` has the following options set:

- fields: `(('name', 'inet:email:header:name'), ('value', 'str'))`

inet:email:header:name

An email header name. The `inet:email:header:name` type is derived from the base type: `str`.

An example of `inet:email:header:name`:

- `subject`

The type `inet:email:header:name` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:email:message

An individual email message delivered to an inbox. The `inet:email:message` type is derived from the base type: `guid`.

inet:email:message:attachment

A file which was attached to an email message. The `inet:email:message:attachment` type is derived from the base type: `comp`.

The type `inet:email:message:attachment` has the following options set:

- fields: `(('message', 'inet:email:message'), ('file', 'file:bytes'))`

inet:email:message:link

A url/link embedded in an email message. The `inet:email:message:link` type is derived from the base type: `comp`.

The type `inet:email:message:link` has the following options set:

- fields: `(('message', 'inet:email:message'), ('url', 'inet:url'))`

inet:flow

An individual network connection between a given source and destination. The `inet:flow` type is derived from the base type: `guid`.

inet:group

A group name string. The `inet:group` type is derived from the base type: `str`.

The type `inet:group` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:http:header

An HTTP protocol header key/value. The `inet:http:header` type is derived from the base type: `comp`.

The type `inet:http:header` has the following options set:

- fields: `(('name', 'inet:http:header:name'), ('value', 'str'))`

inet:http:header:name

The base string type. The `inet:http:header:name` type is derived from the base type: `str`.

The type `inet:http:header:name` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:http:param

An HTTP request path query parameter. The `inet:http:param` type is derived from the base type: `comp`.

The type `inet:http:param` has the following options set:

- fields: `(('name', 'str'), ('value', 'str'))`

inet:http:request

A single HTTP request. The `inet:http:request` type is derived from the base type: `guid`.

inet:http:request:header

An HTTP request header. The `inet:http:request:header` type is derived from the base type: `inet:http:header`.

The type `inet:http:request:header` has the following options set:

- fields: `(('name', 'inet:http:header:name'), ('value', 'str'))`

inet:http:response:header

An HTTP response header. The `inet:http:response:header` type is derived from the base type: `inet:http:header`.

The type `inet:http:response:header` has the following options set:

- fields: `(('name', 'inet:http:header:name'), ('value', 'str'))`

inet:http:session

An HTTP session. The `inet:http:session` type is derived from the base type: `guid`.

inet:iface

A network interface with a set of associated protocol addresses. The `inet:iface` type is derived from the base type: `guid`.

inet:mac

A 48-bit Media Access Control (MAC) address. The `inet:mac` type is derived from the base type: `str`.

An example of `inet:mac`:

- `aa:bb:cc:dd:ee:ff`

The type `inet:mac` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^([0-9a-f]{2}[:]){5}([0-9a-f]{2})$`

- replace: ()
- strip: False

inet:net4

An IPv4 address range. The inet:net4 type is derived from the base type: inet:ipv4range.

An example of inet:net4:

- (1.2.3.4, 1.2.3.20)

The type inet:net4 has the following options set:

- type: ('inet:ipv4', {})

inet:net6

An IPv6 address range. The inet:net6 type is derived from the base type: inet:ipv6range.

An example of inet:net6:

- ('ff::00', 'ff::30')

The type inet:net6 has the following options set:

- type: ('inet:ipv6', {})

inet:passwd

A password string. The inet:passwd type is derived from the base type: str.

The type inet:passwd has the following options set:

- globsuffix: False
- lower: False
- onespace: False
- regex: None
- replace: ()
- strip: False

inet:port

A network port. The inet:port type is derived from the base type: int.

An example of inet:port:

- 80

The type inet:port has the following options set:

- enums:strict: True
- fmt: %d
- ismax: False
- ismin: False

- max: 65535
- min: 0
- signed: True
- size: 8

inet:proto

A network protocol name. The inet:proto type is derived from the base type: str.

The type inet:proto has the following options set:

- globsuffix: False
- lower: True
- onespace: False
- regex: ^[a-z0-9+-]+$
- replace: ()
- strip: False

inet:search:query

An instance of a search query issued to a search engine. The inet:search:query type is derived from the base type: guid.

inet:search:result

A single result from a web search. The inet:search:result type is derived from the base type: guid.

inet:server

A network server address. The inet:server type is derived from the base type: inet:addr.

An example of inet:server:

- tcp://1.2.3.4:80

The type inet:server has the following options set:

- globsuffix: False
- lower: False
- onespace: False
- regex: None
- replace: ()
- strip: False

inet:servfile

A file hosted on a server for access over a network protocol. The inet:servfile type is derived from the base type: comp.

The type inet:servfile has the following options set:

- fields: (('server', 'inet:server'), ('file', 'file:bytes'))

inet:ssl:cert

Deprecated. Please use inet:tls:servercert or inet:tls:clientcert. The inet:ssl:cert type is derived from the base type: comp.

The type inet:ssl:cert has the following options set:

- fields: (('server', 'inet:server'), ('file', 'file:bytes'))

inet:ssl:jarmhash

A TLS JARM fingerprint hash. The inet:ssl:jarmhash type is derived from the base type: str.

The type inet:ssl:jarmhash has the following options set:

- globsuffix: False
- lower: True
- onespace: False
- regex: ^(?<ciphers>[0-9a-f]{30})(?<extensions>[0-9a-f]{32})$
- replace: ()
- strip: True

inet:ssl:jarmsample

A JARM hash sample taken from a server. The inet:ssl:jarmsample type is derived from the base type: comp.

The type inet:ssl:jarmsample has the following options set:

- fields: (('server', 'inet:server'), ('jarmhash', 'inet:ssl:jarmhash'))

inet:tls:clientcert

An x509 certificate sent by a client for TLS. The inet:tls:clientcert type is derived from the base type: comp.

An example of inet:tls:clientcert:

- (1.2.3.4:443, 3fdf364e081c14997b291852d1f23868)

The type inet:tls:clientcert has the following options set:

- fields: (('client', 'inet:client'), ('cert', 'crypto:x509:cert'))

inet:tls:handshake

An instance of a TLS handshake between a server and client. The `inet:tls:handshake` type is derived from the base type: `guid`.

inet:tls:ja3:sample

A JA3 sample taken from a client. The `inet:tls:ja3:sample` type is derived from the base type: `comp`.

The type `inet:tls:ja3:sample` has the following options set:

- fields: `(('client', 'inet:client'), ('ja3', 'hash:md5'))`

inet:tls:ja3s:sample

A JA3 sample taken from a server. The `inet:tls:ja3s:sample` type is derived from the base type: `comp`.

The type `inet:tls:ja3s:sample` has the following options set:

- fields: `(('server', 'inet:server'), ('ja3s', 'hash:md5'))`

inet:tls:servercert

An x509 certificate sent by a server for TLS. The `inet:tls:servercert` type is derived from the base type: `comp`.

An example of `inet:tls:servercert`:

- `(1.2.3.4:443, c7437790af01ae1bb2f8f3b684c70bf8)`

The type `inet:tls:servercert` has the following options set:

- fields: `(('server', 'inet:server'), ('cert', 'crypto:x509:cert'))`

inet:tunnel

A specific sequence of hosts forwarding connections such as a VPN or proxy. The `inet:tunnel` type is derived from the base type: `guid`.

inet:tunnel:type:taxonomy

A taxonomy of network tunnel types. The `inet:tunnel:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `inet:tunnel:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:url:mirror

A URL mirror site. The `inet:url:mirror` type is derived from the base type: `comp`.

The type `inet:url:mirror` has the following options set:

- fields: `(('of', 'inet:url'), ('at', 'inet:url'))`

inet:urlfile

A file hosted at a specific Universal Resource Locator (URL). The `inet:urlfile` type is derived from the base type: `comp`.

The type `inet:urlfile` has the following options set:

- fields: `(('url', 'inet:url'), ('file', 'file:bytes'))`

inet:urlredir

A URL that redirects to another URL, such as via a URL shortening service or an HTTP 302 response. The `inet:urlredir` type is derived from the base type: `comp`.

An example of `inet:urlredir`:

- `(http://foo.com/,http://bar.com/)`

The type `inet:urlredir` has the following options set:

- fields: `(('src', 'inet:url'), ('dst', 'inet:url'))`

inet:user

A username string. The `inet:user` type is derived from the base type: `str`.

The type `inet:user` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

inet:web:acct

An account with a given Internet-based site or service. The `inet:web:acct` type is derived from the base type: `comp`.

An example of `inet:web:acct`:

- `twitter.com/invisig0th`

The type `inet:web:acct` has the following options set:

- fields: `(('site', 'inet:fqdn'), ('user', 'inet:user'))`
- sepr: `/`

inet:web:action

An instance of an account performing an action at an Internet-based site or service. The `inet:web:action` type is derived from the base type: `guid`.

inet:web:attachment

An instance of a file being sent to a web service by an account. The `inet:web:attachment` type is derived from the base type: `guid`.

inet:web:channel

A channel within a web service or instance such as slack or discord. The `inet:web:channel` type is derived from the base type: `guid`.

inet:web:chprofile

A change to a web account. Used to capture historical properties associated with an account, as opposed to current data in the inet:web:acct node. The `inet:web:chprofile` type is derived from the base type: `guid`.

inet:web:file

A file posted by a web account. The `inet:web:file` type is derived from the base type: `comp`.

The type `inet:web:file` has the following options set:

- fields: `(('acct', 'inet:web:acct'), ('file', 'file:bytes'))`

inet:web:follows

A web account follows or is connected to another web account. The `inet:web:follows` type is derived from the base type: `comp`.

The type `inet:web:follows` has the following options set:

- fields: `(('follower', 'inet:web:acct'), ('followee', 'inet:web:acct'))`

inet:web:group

A group hosted within or registered with a given Internet-based site or service. The `inet:web:group` type is derived from the base type: `comp`.

An example of `inet:web:group`:

- `somesite.com/mycoolgroup`

The type `inet:web:group` has the following options set:

- fields: `(('site', 'inet:fqdn'), ('id', 'inet:group'))`
- sepr: `/`

inet:web:hashtag

A hashtag used in a web post. The `inet:web:hashtag` type is derived from the base type: `str`.

The type `inet:web:hashtag` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^#[\w]+$`
- replace: `()`
- strip: `False`

inet:web:instance

An instance of a web service such as slack or discord. The `inet:web:instance` type is derived from the base type: `guid`.

inet:web:logon

An instance of an account authenticating to an Internet-based site or service. The `inet:web:logon` type is derived from the base type: `guid`.

inet:web:memb

Deprecated. Please use inet:web:member. The `inet:web:memb` type is derived from the base type: `comp`.

The type `inet:web:memb` has the following options set:

- fields: `(('acct', 'inet:web:acct'), ('group', 'inet:web:group'))`

inet:web:member

Represents a web account membership in a channel or group. The `inet:web:member` type is derived from the base type: `guid`.

inet:web:mesg

A message sent from one web account to another web account or channel. The `inet:web:mesg` type is derived from the base type: `comp`.

An example of `inet:web:mesg`:

- `((twitter.com, invisig0th), (twitter.com, gobbles), 20041012130220)`

The type `inet:web:mesg` has the following options set:

- fields: `(('from', 'inet:web:acct'), ('to', 'inet:web:acct'), ('time', 'time'))`

inet:web:post

A post made by a web account. The `inet:web:post` type is derived from the base type: `guid`.

inet:web:post:link

A link contained within post text. The `inet:web:post:link` type is derived from the base type: `guid`.

inet:whois:contact

An individual contact from a domain whois record. The `inet:whois:contact` type is derived from the base type: `comp`.

The type `inet:whois:contact` has the following options set:

- fields: `(('rec', 'inet:whois:rec'), ('type', ('str', {'lower': True})))`

inet:whois:email

An email address associated with an FQDN via whois registration text. The `inet:whois:email` type is derived from the base type: `comp`.

The type `inet:whois:email` has the following options set:

- fields: `(('fqdn', 'inet:fqdn'), ('email', 'inet:email'))`

inet:whois:ipcontact

An individual contact from an IP block record. The `inet:whois:ipcontact` type is derived from the base type: `guid`.

inet:whois:ipquery

Query details used to retrieve an IP record. The `inet:whois:ipquery` type is derived from the base type: `guid`.

inet:whois:iprec

An IPv4/IPv6 block registration record. The `inet:whois:iprec` type is derived from the base type: `guid`.

inet:whois:rar

A domain registrar. The `inet:whois:rar` type is derived from the base type: `str`.

An example of `inet:whois:rar`:

- `godaddy, inc.`

The type `inet:whois:rar` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`

- regex: None
- replace: ()
- strip: False

inet:whois:rec

A domain whois record. The inet:whois:rec type is derived from the base type: comp.

The type inet:whois:rec has the following options set:

- fields: (('fqdn', 'inet:fqdn'), ('asof', 'time'))

inet:whois:recns

A nameserver associated with a domain whois record. The inet:whois:recns type is derived from the base type: comp.

The type inet:whois:recns has the following options set:

- fields: (('ns', 'inet:fqdn'), ('rec', 'inet:whois:rec'))

inet:whois:reg

A domain registrant. The inet:whois:reg type is derived from the base type: str.

An example of inet:whois:reg:

- woot hostmaster

The type inet:whois:reg has the following options set:

- globsuffix: False
- lower: True
- onespace: False
- regex: None
- replace: ()
- strip: False

inet:whois:regid

The registry unique identifier of the registration record. The inet:whois:regid type is derived from the base type: str.

An example of inet:whois:regid:

- NET-10-0-0-0-1

The type inet:whois:regid has the following options set:

- globsuffix: False
- lower: False
- onespace: False

- regex: `None`
- replace: `()`
- strip: `False`

inet:wifi:ap

An SSID/MAC address combination for a wireless access point. The `inet:wifi:ap` type is derived from the base type: `comp`.

The type `inet:wifi:ap` has the following options set:

- fields: `(('ssid', 'inet:wifi:ssid'), ('bssid', 'inet:mac'))`

inet:wifi:ssid

A WiFi service set identifier (SSID) name. The `inet:wifi:ssid` type is derived from the base type: `str`.

An example of `inet:wifi:ssid`:

- `The Vertex Project`

The type `inet:wifi:ssid` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

iso:3166:cc

An ISO 3166 2 digit country code. The `iso:3166:cc` type is derived from the base type: `str`.

The type `iso:3166:cc` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z]{2}$`
- replace: `()`
- strip: `False`

iso:oid

An ISO Object Identifier string. The `iso:oid` type is derived from the base type: `str`.

The type `iso:oid` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^([0-2])((\.0)|(\.[1-9][0-9]*))*$`
- replace: `()`
- strip: `False`

it:account

A GUID that represents an account on a host or network. The `it:account` type is derived from the base type: `guid`.

it:adid

An advertising identification string. The `it:adid` type is derived from the base type: `str`.

The type `it:adid` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

it:app:snort:hit

An instance of a snort rule hit. The `it:app:snort:hit` type is derived from the base type: `guid`.

it:app:snort:rule

A snort rule. The `it:app:snort:rule` type is derived from the base type: `guid`.

it:app:yara:match

A YARA rule match to a file. The `it:app:yara:match` type is derived from the base type: `comp`.

The type `it:app:yara:match` has the following options set:

- fields: `(('rule', 'it:app:yara:rule'), ('file', 'file:bytes'))`

it:app:yara:procmatch

An instance of a YARA rule match to a process. The `it:app:yara:procmatch` type is derived from the base type: `guid`.

it:app:yara:rule

A YARA rule unique identifier. The `it:app:yara:rule` type is derived from the base type: `guid`.

it:auth:passwdhash

An instance of a password hash. The `it:auth:passwdhash` type is derived from the base type: `guid`.

it:av:filehit

Deprecated. Please use it:av:scan:result. The `it:av:filehit` type is derived from the base type: `comp`.

The type `it:av:filehit` has the following options set:

- fields: `(('file', 'file:bytes'), ('sig', 'it:av:sig'))`

it:av:prochit

Deprecated. Please use it:av:scan:result. The `it:av:prochit` type is derived from the base type: `guid`.

it:av:scan:result

The result of running an antivirus scanner. The `it:av:scan:result` type is derived from the base type: `guid`.

it:av:sig

Deprecated. Please use it:av:scan:result. The `it:av:sig` type is derived from the base type: `comp`.

The type `it:av:sig` has the following options set:

- fields: `(('soft', 'it:prod:soft'), ('name', 'it:av:signame'))`

it:av:signame

An antivirus signature name. The `it:av:signame` type is derived from the base type: `str`.

The type `it:av:signame` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:cmd

A unique command-line string. The `it:cmd` type is derived from the base type: `str`.

An example of `it:cmd`:

- `foo.exe --dostuff bar`

The type `it:cmd` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

it:dev:int

A developer selected integer constant. The `it:dev:int` type is derived from the base type: `int`.

The type `it:dev:int` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

it:dev:mutex

A string representing a mutex. The `it:dev:mutex` type is derived from the base type: `str`.

The type `it:dev:mutex` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:dev:pipe

A string representing a named pipe. The `it:dev:pipe` type is derived from the base type: `str`.

The type `it:dev:pipe` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:dev:regkey

A Windows registry key. The `it:dev:regkey` type is derived from the base type: `str`.

An example of `it:dev:regkey`:

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`

The type `it:dev:regkey` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:dev:regval

A Windows registry key/value pair. The `it:dev:regval` type is derived from the base type: `guid`.

it:dev:repo

A version control system instance. The `it:dev:repo` type is derived from the base type: `guid`.

it:dev:repo:branch

A branch in a version control system instance. The `it:dev:repo:branch` type is derived from the base type: `guid`.

it:dev:repo:commit

A commit to a repository. The `it:dev:repo:commit` type is derived from the base type: `guid`.

it:dev:repo:diff

A diff of a file being applied in a single commit. The `it:dev:repo:diff` type is derived from the base type: `guid`.

it:dev:repo:diff:comment

A comment on a diff in a repository. The `it:dev:repo:diff:comment` type is derived from the base type: `guid`.

it:dev:repo:issue

An issue raised in a repository. The `it:dev:repo:issue` type is derived from the base type: `guid`.

it:dev:repo:issue:comment

A comment on an issue in a repository. The `it:dev:repo:issue:comment` type is derived from the base type: `guid`.

it:dev:repo:issue:label

A label applied to a repository issue. The `it:dev:repo:issue:label` type is derived from the base type: `guid`.

it:dev:repo:label

A developer selected label. The `it:dev:repo:label` type is derived from the base type: `guid`.

it:dev:repo:remote

A remote repo that is tracked for changes/branches/etc. The `it:dev:repo:remote` type is derived from the base type: `guid`.

it:dev:repo:type:taxonomy

A version control system type taxonomy. The `it:dev:repo:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `it:dev:repo:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:dev:str

A developer selected string. The `it:dev:str` type is derived from the base type: `str`.

The type `it:dev:str` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:domain

A logical boundary of authentication and configuration such as a windows domain. The `it:domain` type is derived from the base type: `guid`.

it:exec:bind

An instance of a host binding a listening port. The `it:exec:bind` type is derived from the base type: `guid`.

it:exec:file:add

An instance of a host adding a file to a filesystem. The `it:exec:file:add` type is derived from the base type: `guid`.

it:exec:file:del

An instance of a host deleting a file from a filesystem. The `it:exec:file:del` type is derived from the base type: `guid`.

it:exec:file:read

An instance of a host reading a file from a filesystem. The `it:exec:file:read` type is derived from the base type: `guid`.

it:exec:file:write

An instance of a host writing a file to a filesystem. The `it:exec:file:write` type is derived from the base type: `guid`.

it:exec:loadlib

A library load event in a process. The `it:exec:loadlib` type is derived from the base type: `guid`.

it:exec:mmap

A memory mapped segment located in a process. The `it:exec:mmap` type is derived from the base type: `guid`.

it:exec:mutex

A mutex created by a process at runtime. The `it:exec:mutex` type is derived from the base type: `guid`.

it:exec:pipe

A named pipe created by a process at runtime. The `it:exec:pipe` type is derived from the base type: `guid`.

it:exec:proc

A process executing on a host. May be an actual (e.g., endpoint) or virtual (e.g., malware sandbox) host. The `it:exec:proc` type is derived from the base type: `guid`.

it:exec:query

An instance of an executed query. The `it:exec:query` type is derived from the base type: `guid`.

it:exec:reg:del

An instance of a host deleting a registry key. The `it:exec:reg:del` type is derived from the base type: `guid`.

it:exec:reg:get

An instance of a host getting a registry key. The `it:exec:reg:get` type is derived from the base type: `guid`.

it:exec:reg:set

An instance of a host creating or setting a registry key. The `it:exec:reg:set` type is derived from the base type: `guid`.

it:exec:thread

A thread executing in a process. The `it:exec:thread` type is derived from the base type: `guid`.

it:exec:url

An instance of a host requesting a URL. The `it:exec:url` type is derived from the base type: `guid`.

it:fs:file

A file on a host. The `it:fs:file` type is derived from the base type: `guid`.

it:group

A GUID that represents a group on a host or network. The `it:group` type is derived from the base type: `guid`.

it:host

A GUID that represents a host or system. The `it:host` type is derived from the base type: `guid`.

it:hostname

The name of a host or system. The `it:hostname` type is derived from the base type: `str`.

The type `it:hostname` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`

- replace: `()`
- strip: `True`

it:hostsoft

A version of a software product which is present on a given host. The `it:hostsoft` type is derived from the base type: `comp`.

The type `it:hostsoft` has the following options set:

- fields: `(('host', 'it:host'), ('softver', 'it:prod:softver'))`

it:hosturl

A url hosted on or served by a host or system. The `it:hosturl` type is derived from the base type: `comp`.

The type `it:hosturl` has the following options set:

- fields: `(('host', 'it:host'), ('url', 'inet:url'))`

it:log:event

A GUID representing an individual log event. The `it:log:event` type is derived from the base type: `guid`.

it:log:event:type:taxonomy

A taxonomy of log event types. The `it:log:event:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `it:log:event:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:logon

A GUID that represents an individual logon/logoff event. The `it:logon` type is derived from the base type: `guid`.

it:mitre:attack:campaign

A MITRE ATT&CK Campaign ID. The `it:mitre:attack:campaign` type is derived from the base type: `str`.

An example of `it:mitre:attack:campaign`:

- `C0028`

The type `it:mitre:attack:campaign` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^C[0-9]{4}$`
- replace: `()`
- strip: `False`

it:mitre:attack:flow

A MITRE ATT&CK Flow diagram. The `it:mitre:attack:flow` type is derived from the base type: `guid`.

it:mitre:attack:group

A MITRE ATT&CK Group ID. The `it:mitre:attack:group` type is derived from the base type: `str`.

An example of `it:mitre:attack:group`:

- `G0100`

The type `it:mitre:attack:group` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^G[0-9]{4}$`
- replace: `()`
- strip: `False`

it:mitre:attack:matrix

An enumeration of ATT&CK matrix values. The `it:mitre:attack:matrix` type is derived from the base type: `str`.

An example of `it:mitre:attack:matrix`:

- `enterprise`

The type `it:mitre:attack:matrix` has the following options set:

- enums:

valu
enterprise
mobile
ics

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:mitre:attack:mitigation

A MITRE ATT&CK Mitigation ID. The `it:mitre:attack:mitigation` type is derived from the base type: `str`.

An example of `it:mitre:attack:mitigation`:

- `M1036`

The type `it:mitre:attack:mitigation` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^M[0-9]{4}$`
- replace: `()`
- strip: `False`

it:mitre:attack:software

A MITRE ATT&CK Software ID. The `it:mitre:attack:software` type is derived from the base type: `str`.

An example of `it:mitre:attack:software`:

- `S0154`

The type `it:mitre:attack:software` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^S[0-9]{4}$`
- replace: `()`

- strip: `False`

it:mitre:attack:status

A MITRE ATT&CK element status. The `it:mitre:attack:status` type is derived from the base type: `str`.

An example of `it:mitre:attack:status`:

- `current`

The type `it:mitre:attack:status` has the following options set:

- enums:

valu
current
deprecated
withdrawn

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:mitre:attack:tactic

A MITRE ATT&CK Tactic ID. The `it:mitre:attack:tactic` type is derived from the base type: `str`.

An example of `it:mitre:attack:tactic`:

- `TA0040`

The type `it:mitre:attack:tactic` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^TA[0-9]{4}$`
- replace: `()`
- strip: `False`

it:mitre:attack:technique

A MITRE ATT&CK Technique ID. The `it:mitre:attack:technique` type is derived from the base type: `str`.

An example of `it:mitre:attack:technique`:

- `T1548`

The type `it:mitre:attack:technique` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^T[0-9]{4}(.[0-9]{3})?$`
- replace: `()`
- strip: `False`

it:network

A GUID that represents a logical network. The `it:network` type is derived from the base type: `guid`.

it:os:android:aaid

An android advertising identification string. The `it:os:android:aaid` type is derived from the base type: `it:adid`.

The type `it:os:android:aaid` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

it:os:android:ibroadcast

The given software broadcasts the given Android intent. The `it:os:android:ibroadcast` type is derived from the base type: `comp`.

The type `it:os:android:ibroadcast` has the following options set:

- fields: `(('app', 'it:prod:soft'), ('intent', 'it:os:android:intent'))`

it:os:android:ilisten

The given software listens for an android intent. The `it:os:android:ilisten` type is derived from the base type: `comp`.

The type `it:os:android:ilisten` has the following options set:

- fields: `(('app', 'it:prod:soft'), ('intent', 'it:os:android:intent'))`

it:os:android:intent

An android intent string. The `it:os:android:intent` type is derived from the base type: `str`.

The type `it:os:android:intent` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:os:android:perm

An android permission string. The `it:os:android:perm` type is derived from the base type: `str`.

The type `it:os:android:perm` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:os:android:reqperm

The given software requests the android permission. The `it:os:android:reqperm` type is derived from the base type: `comp`.

The type `it:os:android:reqperm` has the following options set:

- fields: `(('app', 'it:prod:soft'), ('perm', 'it:os:android:perm'))`

it:os:ios:idfa

An iOS advertising identification string. The `it:os:ios:idfa` type is derived from the base type: `it:adid`.

The type `it:os:ios:idfa` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

it:os:windows:sid

A Microsoft Windows Security Identifier. The `it:os:windows:sid` type is derived from the base type: `str`.

An example of `it:os:windows:sid`:

- `S-1-5-21-1220945662-1202665555-839525555-5555`

The type `it:os:windows:sid` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^S-1-(?:\d{1,10}|0x[0-9a-fA-F]{12})(?:-(?:\d+|0x[0-9a-fA-F]{2,}))*$`
- replace: `()`
- strip: `False`

it:prod:component

A specific instance of an it:prod:hardware most often as part of an it:host. The `it:prod:component` type is derived from the base type: `guid`.

it:prod:hardware

A specification for a piece of IT hardware. The `it:prod:hardware` type is derived from the base type: `guid`.

it:prod:hardwaretype

An IT hardware type taxonomy. The `it:prod:hardwaretype` type is derived from the base type: `taxonomy`.

The type `it:prod:hardwaretype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`

- replace: `()`
- strip: `False`

it:prod:soft

A software product. The `it:prod:soft` type is derived from the base type: `guid`.

it:prod:soft:taxonomy

A software type taxonomy. The `it:prod:soft:taxonomy` type is derived from the base type: `taxonomy`.

The type `it:prod:soft:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:prod:softfile

A file is distributed by a specific software version. The `it:prod:softfile` type is derived from the base type: `comp`.

The type `it:prod:softfile` has the following options set:

- fields: `(('soft', 'it:prod:softver'), ('file', 'file:bytes'))`

it:prod:softid

An identifier issued to a given host by a specific software application. The `it:prod:softid` type is derived from the base type: `guid`.

it:prod:softlib

A software version contains a library software version. The `it:prod:softlib` type is derived from the base type: `comp`.

The type `it:prod:softlib` has the following options set:

- fields: `(('soft', 'it:prod:softver'), ('lib', 'it:prod:softver'))`

it:prod:softname

A software product name. The `it:prod:softname` type is derived from the base type: `str`.

The type `it:prod:softname` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

it:prod:softos

The software version is known to be compatible with the given os software version. The `it:prod:softos` type is derived from the base type: `comp`.

The type `it:prod:softos` has the following options set:

- fields: `(('soft', 'it:prod:softver'), ('os', 'it:prod:softver'))`

it:prod:softreg

A registry entry is created by a specific software version. The `it:prod:softreg` type is derived from the base type: `comp`.

The type `it:prod:softreg` has the following options set:

- fields: `(('softver', 'it:prod:softver'), ('regval', 'it:dev:regval'))`

it:prod:softver

A specific version of a software product. The `it:prod:softver` type is derived from the base type: `guid`.

it:query

A unique query string. The `it:query` type is derived from the base type: `str`.

The type `it:query` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

it:reveng:filefunc

An instance of a function in an executable. The `it:reveng:filefunc` type is derived from the base type: `comp`.

The type `it:reveng:filefunc` has the following options set:

- fields: `(('file', 'file:bytes'), ('function', 'it:reveng:function'))`

it:reveng:funcstr

A reference to a string inside a function. The `it:reveng:funcstr` type is derived from the base type: `comp`.

The type `it:reveng:funcstr` has the following options set:

- fields: `(('function', 'it:reveng:function'), ('string', 'str'))`

it:reveng:function

A function inside an executable. The `it:reveng:function` type is derived from the base type: `guid`.

it:reveng:impfunc

A function from an imported library. The `it:reveng:impfunc` type is derived from the base type: `str`.

The type `it:reveng:impfunc` has the following options set:

- globsuffix: `False`
- lower: `1`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

it:screenshot

A screenshot of a host. The `it:screenshot` type is derived from the base type: `guid`.

it:sec:c2:config

An extracted C2 config from an executable. The `it:sec:c2:config` type is derived from the base type: `guid`.

it:sec:cve

A vulnerability as designated by a Common Vulnerabilities and Exposures (CVE) number. The `it:sec:cve` type is derived from the base type: `str`.

An example of `it:sec:cve`:

- `cve-2012-0158`

The type `it:sec:cve` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `(?i)^CVE-[0-9]{4}-[0-9]{4,}$`
- replace: `(('‑', '-'), ('‒', '-'), ('–', '-'), ('—', '-'))`
- strip: `False`

it:sec:cwe

NIST NVD Common Weaknesses Enumeration Specification. The `it:sec:cwe` type is derived from the base type: `str`.

An example of `it:sec:cwe`:

- `CWE-120`

The type `it:sec:cwe` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^CWE-[0-9]{1,8}$`
- replace: `()`
- strip: `False`

it:sec:metrics

A node used to track metrics of an organization's infosec program. The `it:sec:metrics` type is derived from the base type: `guid`.

it:sec:stix:bundle

A STIX bundle. The `it:sec:stix:bundle` type is derived from the base type: `guid`.

it:sec:stix:indicator

A STIX indicator pattern. The `it:sec:stix:indicator` type is derived from the base type: `guid`.

it:sec:tlp

The US CISA Traffic-Light-Protocol used to designate information sharing boundaries. The `it:sec:tlp` type is derived from the base type: `int`.

An example of `it:sec:tlp`:

- `green`

The type `it:sec:tlp` has the following options set:

- enums:

int	valu
10	clear
20	green
30	amber
40	amber-strict
50	red

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

it:sec:vuln:scan

An instance of running a vulnerability scan. The `it:sec:vuln:scan` type is derived from the base type: `guid`.

it:sec:vuln:scan:result

A vulnerability scan result for an asset. The `it:sec:vuln:scan:result` type is derived from the base type: `guid`.

lang:code

An optionally 2 part language code. The `lang:code` type is derived from the base type: `str`.

An example of `lang:code`:

- `pt.br`

The type `lang:code` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z]{2}(.[a-z]{2})?$`
- replace: `()`
- strip: `False`

lang:idiom

Deprecated. Please use lang:translation. The `lang:idiom` type is derived from the base type: `str`.

The type `lang:idiom` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

lang:language

A specific written or spoken language. The `lang:language` type is derived from the base type: `guid`.

lang:name

A name used to refer to a language. The `lang:name` type is derived from the base type: `str`.

The type `lang:name` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`

- replace: ()
- strip: False

lang:trans

Deprecated. Please use lang:translation. The `lang:trans` type is derived from the base type: `str`.

The type `lang:trans` has the following options set:

- globsuffix: False
- lower: False
- onespace: False
- regex: None
- replace: ()
- strip: False

lang:translation

A translation of text from one language to another. The `lang:translation` type is derived from the base type: `guid`.

mass

A mass which converts to grams as a base unit. The `mass` type is derived from the base type: `hugenum`.

The type `mass` has the following options set:

- modulo: None
- units: `{'µg': '0.000001', 'microgram': '0.000001', 'micrograms': '0.000001', 'mg': '0.001', 'milligram': '0.001', 'milligrams': '0.001', 'g': '1', 'grams': '1', 'kg': '1000', 'kilogram': '1000', 'kilograms': '1000', 'lb': '453.592', 'lbs': '453.592', 'pound': '453.592', 'pounds': '453.592', 'stone': '6350.29'}`

mat:item

A GUID assigned to a material object. The `mat:item` type is derived from the base type: `guid`.

mat:itemimage

The base type for compound node fields. The `mat:itemimage` type is derived from the base type: `comp`.

The type `mat:itemimage` has the following options set:

- fields: `(('item', 'mat:item'), ('file', 'file:bytes'))`

mat:spec

A GUID assigned to a material specification. The `mat:spec` type is derived from the base type: `guid`.

mat:specimage

The base type for compound node fields. The `mat:specimage` type is derived from the base type: `comp`.

The type `mat:specimage` has the following options set:

- fields: `(('spec', 'mat:spec'), ('file', 'file:bytes'))`

mat:type

A taxonomy of material item/specification types. The `mat:type` type is derived from the base type: `taxonomy`.

The type `mat:type` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

media:news

A GUID for a news article or report. The `media:news` type is derived from the base type: `guid`.

media:news:taxonomy

A taxonomy of types or sources of news. The `media:news:taxonomy` type is derived from the base type: `taxonomy`.

The type `media:news:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

media:topic

A topic string. The `media:topic` type is derived from the base type: `str`.

The type `media:topic` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

meta:event

An analytically relevant event in a curated timeline. The `meta:event` type is derived from the base type: `guid`.

meta:event:taxonomy

A taxonomy of event types for meta:event nodes. The `meta:event:taxonomy` type is derived from the base type: `taxonomy`.

The type `meta:event:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

meta:note

An analyst note about nodes linked with -(about)> edges. The `meta:note` type is derived from the base type: `guid`.

meta:note:type:taxonomy

An analyst note type taxonomy. The `meta:note:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `meta:note:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

meta:priority

A generic priority enumeration. The `meta:priority` type is derived from the base type: `int`.

The type `meta:priority` has the following options set:

- enums:

int	valu
0	none
10	lowest
20	low
30	medium
40	high
50	highest

- enums:strict: `False`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

meta:rule

A generic rule linked to matches with -(matches)> edges. The `meta:rule` type is derived from the base type: `guid`.

meta:ruleset

A set of rules linked with -(has)> edges. The `meta:ruleset` type is derived from the base type: `guid`.

meta:seen

Annotates that the data in a node was obtained from or observed by a given source. The `meta:seen` type is derived from the base type: `comp`.

The type `meta:seen` has the following options set:

- fields: `(('source', 'meta:source'), ('node', 'ndef'))`

meta:severity

A generic severity enumeration. The `meta:severity` type is derived from the base type: `int`.

The type `meta:severity` has the following options set:

- enums:

int	valu
0	none
10	lowest
20	low
30	medium
40	high
50	highest

- enums:strict: `False`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

meta:sophistication

A sophistication score with named values: very low, low, medium, high, and very high. The `meta:sophistication` type is derived from the base type: `int`.

The type `meta:sophistication` has the following options set:

- enums:

int	valu
10	very low
20	low
30	medium
40	high
50	very high

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`

- min: None
- signed: True
- size: 8

meta:source

A data source unique identifier. The `meta:source` type is derived from the base type: `guid`.

meta:timeline

A curated timeline of analytically relevant events. The `meta:timeline` type is derived from the base type: `guid`.

meta:timeline:taxonomy

A taxonomy of timeline types for meta:timeline nodes. The `meta:timeline:taxonomy` type is derived from the base type: `taxonomy`.

The type `meta:timeline:taxonomy` has the following options set:

- globsuffix: False
- lower: False
- onespace: False
- regex: None
- replace: ()
- strip: False

ou:alias

An alias for the org GUID. The `ou:alias` type is derived from the base type: `str`.

An example of `ou:alias`:

- `vertexproject`

The type `ou:alias` has the following options set:

- globsuffix: False
- lower: True
- onespace: False
- regex: `^[0-9a-z_]+$`
- replace: ()
- strip: False

ou:attendee

A node representing a person attending a meeting, conference, or event. The `ou:attendee` type is derived from the base type: `guid`.

ou:award

An award issued by an organization. The `ou:award` type is derived from the base type: `guid`.

ou:campaign

Represents an org's activity in pursuit of a goal. The `ou:campaign` type is derived from the base type: `guid`.

ou:campname

A campaign name. The `ou:campname` type is derived from the base type: `str`.

The type `ou:campname` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

ou:camptype

An campaign type taxonomy. The `ou:camptype` type is derived from the base type: `taxonomy`.

The type `ou:camptype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:conference

A conference with a name and sponsoring org. The `ou:conference` type is derived from the base type: `guid`.

ou:conference:attendee

Deprecated. Please use ou:attendee. The `ou:conference:attendee` type is derived from the base type: `comp`.

The type `ou:conference:attendee` has the following options set:

- fields: `(('conference', 'ou:conference'), ('person', 'ps:person'))`

ou:conference:event

A conference event with a name and associated conference. The `ou:conference:event` type is derived from the base type: `guid`.

ou:conference:event:attendee

Deprecated. Please use ou:attendee. The `ou:conference:event:attendee` type is derived from the base type: `comp`.

The type `ou:conference:event:attendee` has the following options set:

- fields: `(('conference', 'ou:conference:event'), ('person', 'ps:person'))`

ou:conflict

Represents a conflict where two or more campaigns have mutually exclusive goals. The `ou:conflict` type is derived from the base type: `guid`.

ou:contest

A competitive event resulting in a ranked set of participants. The `ou:contest` type is derived from the base type: `guid`.

ou:contest:result

The results from a single contest participant. The `ou:contest:result` type is derived from the base type: `comp`.

The type `ou:contest:result` has the following options set:

- fields: `(('contest', 'ou:contest'), ('participant', 'ps:contact'))`

ou:contract

An contract between multiple entities. The `ou:contract` type is derived from the base type: `guid`.

ou:contract:type

A pre-defined set of contract types. The `ou:contract:type` type is derived from the base type: `str`.

The type `ou:contract:type` has the following options set:

- enum: `('nda', 'other', 'grant', 'treaty', 'purchase', 'indemnity', 'partnership')`
- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:contribution

Represents a specific instance of contributing material support to a campaign. The `ou:contribution` type is derived from the base type: `guid`.

ou:conttype

A contract type taxonomy. The `ou:conttype` type is derived from the base type: `taxonomy`.

The type `ou:conttype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:employment

An employment type taxonomy. The `ou:employment` type is derived from the base type: `taxonomy`.

An example of `ou:employment`:

- `fulltime.salary`

The type `ou:employment` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`

- regex: None
- replace: ()
- strip: False

ou:goal

An assessed or stated goal which may be abstract or org specific. The ou:goal type is derived from the base type: guid.

ou:goal:type:taxonomy

A taxonomy of goal types. The ou:goal:type:taxonomy type is derived from the base type: taxonomy.

The type ou:goal:type:taxonomy has the following options set:

- globsuffix: False
- lower: False
- onespace: False
- regex: None
- replace: ()
- strip: False

ou:goalname

A goal name. The ou:goalname type is derived from the base type: str.

The type ou:goalname has the following options set:

- globsuffix: False
- lower: True
- onespace: True
- regex: None
- replace: ()
- strip: False

ou:hasalias

The knowledge that an organization has an alias. The ou:hasalias type is derived from the base type: comp.

The type ou:hasalias has the following options set:

- fields: (('org', 'ou:org'), ('alias', 'ou:alias'))

ou:hasgoal

Deprecated. Please use ou:org:goals. The `ou:hasgoal` type is derived from the base type: `comp`.

The type `ou:hasgoal` has the following options set:

- fields: `(('org', 'ou:org'), ('goal', 'ou:goal'))`

ou:id:number

A unique id number issued by a specific organization. The `ou:id:number` type is derived from the base type: `comp`.

The type `ou:id:number` has the following options set:

- fields: `(('type', 'ou:id:type'), ('value', 'ou:id:value'))`

ou:id:type

A type of id number issued by an org. The `ou:id:type` type is derived from the base type: `guid`.

ou:id:update

A status update to an org:id:number. The `ou:id:update` type is derived from the base type: `guid`.

ou:id:value

The value of an org:id:number. The `ou:id:value` type is derived from the base type: `str`.

The type `ou:id:value` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

ou:industry

An industry classification type. The `ou:industry` type is derived from the base type: `guid`.

ou:industry:type:taxonomy

An industry type taxonomy. The `ou:industry:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `ou:industry:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:industryname

The name of an industry. The `ou:industryname` type is derived from the base type: `str`.

The type `ou:industryname` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

ou:isic

An International Standard Industrial Classification of All Economic Activities (ISIC) code. The `ou:isic` type is derived from the base type: `str`.

An example of `ou:isic`:

- `C1393`

The type `ou:isic` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[A-Z]([0-9]{2}[0-9]{0,2})?$`
- replace: `()`
- strip: `False`

ou:jobtitle

A title for a position within an org. The `ou:jobtitle` type is derived from the base type: `str`.

The type `ou:jobtitle` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

ou:jobtype

A taxonomy of job types. The `ou:jobtype` type is derived from the base type: `taxonomy`.

An example of `ou:jobtype`:

- `it.dev.python`

The type `ou:jobtype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:meet

An informal meeting of people which has no title or sponsor. See also: ou:conference. The `ou:meet` type is derived from the base type: `guid`.

ou:meet:attendee

Deprecated. Please use ou:attendee. The `ou:meet:attendee` type is derived from the base type: `comp`.

The type `ou:meet:attendee` has the following options set:

- fields: `(('meet', 'ou:meet'), ('person', 'ps:person'))`

ou:member

Deprecated. Please use ou:position. The `ou:member` type is derived from the base type: `comp`.

The type `ou:member` has the following options set:

- fields: `(('org', 'ou:org'), ('person', 'ps:person'))`

ou:naics

North American Industry Classification System codes and prefixes. The `ou:naics` type is derived from the base type: `str`.

An example of `ou:naics`:

- `541715`

The type `ou:naics` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[1-9][0-9]{1,5}?$`
- replace: `()`
- strip: `True`

ou:name

The name of an organization. This may be a formal name or informal name of the organization. The `ou:name` type is derived from the base type: `str`.

An example of `ou:name`:

- `acme corporation`

The type `ou:name` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

ou:opening

A job/work opening within an org. The `ou:opening` type is derived from the base type: `guid`.

ou:org

A GUID for a human organization such as a company or military unit. The `ou:org` type is derived from the base type: `guid`.

ou:org:has

An org owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time. The `ou:org:has` type is derived from the base type: `comp`.

The type `ou:org:has` has the following options set:

- fields: `(('org', 'ou:org'), ('node', 'ndef'))`

ou:orgnet4

An organization's IPv4 netblock. The `ou:orgnet4` type is derived from the base type: `comp`.

The type `ou:orgnet4` has the following options set:

- fields: `(('org', 'ou:org'), ('net', 'inet:net4'))`

ou:orgnet6

An organization's IPv6 netblock. The `ou:orgnet6` type is derived from the base type: `comp`.

The type `ou:orgnet6` has the following options set:

- fields: `(('org', 'ou:org'), ('net', 'inet:net6'))`

ou:orgtype

An org type taxonomy. The `ou:orgtype` type is derived from the base type: `taxonomy`.

The type `ou:orgtype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:position

A position within an org. May be organized into an org chart. The `ou:position` type is derived from the base type: `guid`.

ou:preso

A webinar, conference talk, or other type of presentation. The `ou:preso` type is derived from the base type: `guid`.

ou:requirement

A specific requirement. The `ou:requirement` type is derived from the base type: `guid`.

ou:role

A named role when participating in an event. The `ou:role` type is derived from the base type: `str`.

An example of `ou:role`:

- `staff`

The type `ou:role` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^\w+$`
- replace: `()`
- strip: `False`

ou:sic

The four digit Standard Industrial Classification Code. The `ou:sic` type is derived from the base type: `str`.

An example of `ou:sic`:

- `0111`

The type `ou:sic` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[0-9]{4}$`
- replace: `()`
- strip: `False`

ou:suborg

Any parent/child relationship between two orgs. May represent ownership, organizational structure, etc. The `ou:suborg` type is derived from the base type: `comp`.

The type `ou:suborg` has the following options set:

- fields: `(('org', 'ou:org'), ('sub', 'ou:org'))`

ou:team

A GUID for a team within an organization. The `ou:team` type is derived from the base type: `guid`.

ou:technique

A specific technique used to achieve a goal. The `ou:technique` type is derived from the base type: `guid`.

ou:technique:taxonomy

An analyst defined taxonomy to classify techniques in different disciplines. The `ou:technique:taxonomy` type is derived from the base type: `taxonomy`.

The type `ou:technique:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ou:user

A user name within an organization. The `ou:user` type is derived from the base type: `comp`.

The type `ou:user` has the following options set:

- fields: `(('org', 'ou:org'), ('user', 'inet:user'))`

ou:vitals

Vital statistics about an org for a given time period. The `ou:vitals` type is derived from the base type: `guid`.

pe:langid

The PE language id. The `pe:langid` type is derived from the base type: `int`.

The type `pe:langid` has the following options set:

- enums:

int	valu
0	neutral
4	zh-Hans
26	hr
127	invariant
1024	default
1025	ar-SA
1026	bg-BG
1027	ca-ES
1029	cs-CZ
1030	da-DK
1031	de-DE
1032	el-GR
1033	en-US
1034	es-ES-traditional
1035	fi-FI
1036	fr-FR
1037	he-IL
1038	hu-HU
1039	is-IS
1040	it-IT
1041	ja-JP
1042	ko-KR
1043	nl-NL
1044	nb-NO
1045	pl-PL
1046	pt-BR
1047	rm-CH
1048	ro-RO
1049	ru-RU
1050	hr-HR
1051	sk-SK
1052	sq-AL
1053	sv-SE
1054	th-TH
1055	tr-TR
1056	ur-PK
1057	id-ID
1058	uk-UA
1059	be-BY
1060	sl-SI
1061	et-EE
1062	lv-LV
1063	lt-LT
1064	tg-TJ

continues on next page

Table 1 – continued from previous page

int	valu
1065	fa-IR
1066	vi-VN
1067	hy-AM
1068	az-AZ-Latin
1069	Basque-Basque
1070	hsb-DE
1071	mk-MK
1074	tn-ZA
1076	xh-ZA
1077	zu-ZA
1078	af-ZA
1079	ka-GE
1080	fo-FO
1081	hi-IN
1082	mt-MT
1083	se-NO
1086	ms-MY
1087	kk-KZ
1088	ky-KG
1089	sw-KE
1090	tk-TM
1091	uz-UZ-Latin
1092	tt-RU
1093	bn-Bangledesh
1094	pa-IN
1095	gu-IN
1096	or-IN
1097	ta-IN
1098	te-IN
1099	kn-IN
1100	ml-IN
1101	as-IN
1102	mr-IN
1103	sa-IN
1104	mn-MN-Cyrllic
1105	bo-CN
1106	cy-GB
1107	kh-KH
1108	lo-LA
1110	gl-ES
1111	kok-IN
1114	syr-SY
1115	si-LK
1116	chr-Cher
1117	iu-CA
1118	am-ET
1121	ne-NP
1122	fy-NL
1123	ps-AF
1124	fil-PH

continues on next page

Table 1 – continued from previous page

int	valu
1125	dv-MV
1128	ha-NG
1130	yo-NG
1131	quz-BO
1132	nso-ZA
1133	ba-RU
1134	lb-LU
1135	kl-GL
1136	ig-NG
1139	ti-ET
1141	haw-US
1144	ii-CN
1146	arn-CL
1148	moh-CA
1150	br-FR
1152	ug-CN
1153	mi-NZ
1154	oc-FR
1155	co-FR
1156	gsw-FR
1157	sah-RU
1158	qut-GT
1159	rw-RW
1160	wo-SN
1164	prs-AF
1170	ku-IQ
2048	sys default
2049	ar-IQ
2051	ca-ES-Valencia
2055	de-CH
2057	en-GB
2058	es-MX
2060	fr-BE
2064	it-CH
2067	nl-BE
2068	no-NO
2070	pt-PT
2074	sr-CS-Latin
2077	sv-FI
2080	ur-IN
2092	az-AZ-Cyrillic
2094	dsb-DE
2098	tn-BW
2107	se-SE
2108	ga-IE
2110	ms-BN
2115	uz-UZ-Cyrillic
2117	bn-IN
2118	pa-PK
2121	ta-LK

continues on next page

Table 1 – continued from previous page

int	valu
2128	mn-MN-Prc
2137	sd-PK
2141	iu-CA-Latin
2143	tzm-DZ
2151	ff-SN
2155	quz-EC
2163	ti-ER
3072	custom default
3073	ar-EG
3076	zh-HK
3079	de-AT
3081	en-AU
3082	es-ES-modern
3084	fr-CA
3098	sr-CS-Cyrillic
3131	se-FI
3179	quz-PE
4096	custom unspecified
4097	ar-LY
4100	zh-SG
4103	de-LU
4105	en-CA
4106	es-GT
4108	fr-CH
4122	hr-BA
4155	smj-NO
5120	ui_custom_default
5121	ar-DZ
5124	zh-MO
5127	de-LI
5129	en-NZ
5130	es-CR
5132	fr-LU
5146	bs-BA-Latin
5179	smj-SE
6145	ar-MA
6153	en-IE
6154	es-PA
6156	fr-MC
6170	sr-code-Latin
6203	sma-NO
7169	ar-TN
7177	en-ZA
7178	es-DO
7194	sr-BA
7227	sma-SE
8193	ar-OM
8201	en-JM
8202	es-VE
8218	bs-BA-Cyrillic

continues on next page

Table 1 – continued from previous page

int	valu
8251	sms-FI
9217	ar-YE
9225	en-029
9226	es-CO
9275	smn-FIl
10241	ar-SY
10249	en-BZ
10250	es-PE
11265	ar-JO
11273	en-TT
11274	es-AR
12289	ar-LB
12297	en-ZW
12298	es-EC
13313	ar-KW
13321	en-PH
13322	es-CL
14337	ar-AE
14346	es-UY
15361	ar-BH
15370	es-PY
16385	ar-QA
16393	en-IN
16394	es-BO
17417	en-MY
17418	es-SV
18441	en-SG
18442	es-HN
19466	es-NI
20490	es-PR
21514	es-US
30746	bs-neutral
31748	zh-Hant
31770	sr-Neutral

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

pe:resource:type

The typecode for the resource. The `pe:resource:type` type is derived from the base type: `int`.

The type `pe:resource:type` has the following options set:

- enums:

int	valu
1	RT_CURSOR
2	RT_BITMAP
3	RT_ICON
4	RT_MENU
5	RT_DIALOG
6	RT_STRING
7	RT_FONTDIR
8	RT_FONT
9	RT_ACCELERATOR
10	RT_RCDATA
11	RT_MESSAGETABLE
12	RT_GROUP_CURSOR
14	RT_GROUP_ICON
16	RT_VERSION
17	RT_DLGINCLUDE
19	RT_PLUGPLAY
20	RT_VXD
21	RT_ANICURSOR
22	RT_ANIICON
23	RT_HTML
24	RT_MANIFEST

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

pol:candidate

A candidate for office in a specific race. The `pol:candidate` type is derived from the base type: `guid`.

pol:country

A GUID for a country. The `pol:country` type is derived from the base type: `guid`.

pol:election

An election involving one or more races for office. The `pol:election` type is derived from the base type: `guid`.

pol:immigration:status

A node which tracks the immigration status of a contact. The `pol:immigration:status` type is derived from the base type: `guid`.

pol:immigration:status:type:taxonomy

A taxonomy of immigration types. The `pol:immigration:status:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `pol:immigration:status:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

pol:iso2

The 2 digit ISO 3166 country code. The `pol:iso2` type is derived from the base type: `str`.

An example of `pol:iso2`:

- `us`

The type `pol:iso2` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z0-9]{2}$`
- replace: `()`
- strip: `False`

pol:iso3

The 3 digit ISO 3166 country code. The `pol:iso3` type is derived from the base type: `str`.

An example of `pol:iso3`:

- `usa`

The type `pol:iso3` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z0-9]{3}$`
- replace: `()`
- strip: `False`

pol:isonum

The ISO integer country code. The `pol:isonum` type is derived from the base type: `int`.

An example of `pol:isonum`:

- `840`

The type `pol:isonum` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

pol:office

An elected or appointed office. The `pol:office` type is derived from the base type: `guid`.

pol:pollingplace

An official place where ballots may be cast for a specific election. The `pol:pollingplace` type is derived from the base type: `guid`.

pol:race

An individual race for office. The `pol:race` type is derived from the base type: `guid`.

pol:term

A term in office held by a specific individual. The `pol:term` type is derived from the base type: `guid`.

pol:vitals

A set of vital statistics about a country. The `pol:vitals` type is derived from the base type: `guid`.

proj:attachment

A file attachment added to a ticket or comment. The `proj:attachment` type is derived from the base type: `guid`.

proj:comment

A user comment on a ticket. The `proj:comment` type is derived from the base type: `guid`.

proj:epic

A collection of tickets related to a topic. The `proj:epic` type is derived from the base type: `guid`.

proj:project

A project in a ticketing system. The `proj:project` type is derived from the base type: `guid`.

proj:project:type:taxonomy

A type taxonomy for projects. The `proj:project:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `proj:project:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

proj:sprint

A timeboxed period to complete a set amount of work. The `proj:sprint` type is derived from the base type: `guid`.

proj:ticket

A ticket in a ticketing system. The `proj:ticket` type is derived from the base type: `guid`.

ps:achievement

An instance of an individual receiving an award. The `ps:achievement` type is derived from the base type: `guid`.

ps:contact

A GUID for a contact info record. The `ps:contact` type is derived from the base type: `guid`.

ps:contact:type:taxonomy

A taxonomy of contact types. The `ps:contact:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `ps:contact:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ps:contactlist

A GUID for a list of associated contacts. The `ps:contactlist` type is derived from the base type: `guid`.

ps:education

A period of education for an individual. The `ps:education` type is derived from the base type: `guid`.

ps:name

An arbitrary, lower spaced string with normalized whitespace. The `ps:name` type is derived from the base type: `str`.

An example of `ps:name`:

- `robert grey`

The type `ps:name` has the following options set:

- globsuffix: `False`
- lower: `True`

- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

ps:person

A GUID for a person. The `ps:person` type is derived from the base type: `guid`.

ps:person:has

A person owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time. The `ps:person:has` type is derived from the base type: `comp`.

The type `ps:person:has` has the following options set:

- fields: `(('person', 'ps:person'), ('node', 'ndef'))`

ps:persona

A GUID for a suspected person. The `ps:persona` type is derived from the base type: `guid`.

ps:persona:has

A persona owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time. The `ps:persona:has` type is derived from the base type: `comp`.

The type `ps:persona:has` has the following options set:

- fields: `(('persona', 'ps:persona'), ('node', 'ndef'))`

ps:proficiency

The assessment that a given contact possesses a specific skill. The `ps:proficiency` type is derived from the base type: `guid`.

ps:skill

A specific skill which a person or organization may have. The `ps:skill` type is derived from the base type: `guid`.

ps:skill:type:taxonomy

A taxonomy of skill types. The `ps:skill:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `ps:skill:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

ps:tokn

A single name element (potentially given or sur). The `ps:tokn` type is derived from the base type: `str`.

An example of `ps:tokn`:

- `robert`

The type `ps:tokn` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

ps:vitals

Statistics and demographic data about a person or contact. The `ps:vitals` type is derived from the base type: `guid`.

ps:workhist

A GUID representing entry in a contact's work history. The `ps:workhist` type is derived from the base type: `guid`.

risk:alert

An instance of an alert which indicates the presence of a risk. The `risk:alert` type is derived from the base type: `guid`.

risk:alert:taxonomy

A taxonomy of alert types. The `risk:alert:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:alert:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:alert:verdict:taxonomy

A taxonomy of verdicts for the origin and validity of the alert. The `risk:alert:verdict:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:alert:verdict:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:attack

An instance of an actor attacking a target. The `risk:attack` type is derived from the base type: `guid`.

risk:attacktype

A taxonomy of attack types. The `risk:attacktype` type is derived from the base type: `taxonomy`.

The type `risk:attacktype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:availability

A taxonomy of availability status values. The `risk:availability` type is derived from the base type: `taxonomy`.

The type `risk:availability` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:compromise

An instance of a compromise and its aggregate impact. The `risk:compromise` type is derived from the base type: `guid`.

risk:compromisetype

A taxonomy of compromise types. The `risk:compromisetype` type is derived from the base type: `taxonomy`.

An example of `risk:compromisetype`:

- `cno.breach`

The type `risk:compromisetype` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:extortion

An event where an attacker attempted to extort a victim. The `risk:extortion` type is derived from the base type: `guid`.

risk:extortion:type:taxonomy

A taxonomy of extortion event types. The `risk:extortion:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:extortion:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:hasvuln

Deprecated. Please use risk:vulnerable. The `risk:hasvuln` type is derived from the base type: `guid`.

risk:leak

An event where information was disclosed without permission. The `risk:leak` type is derived from the base type: `guid`.

risk:leak:type:taxonomy

A taxonomy of leak event types. The `risk:leak:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:leak:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:mitigation

A mitigation for a specific risk:vuln. The `risk:mitigation` type is derived from the base type: `guid`.

risk:technique:masquerade

Represents the assessment that a node is designed to resemble another in order to mislead. The `risk:technique:masquerade` type is derived from the base type: `guid`.

risk:threat

A threat cluster or subgraph of threat activity, as reported by a specific organization. The `risk:threat` type is derived from the base type: `guid`.

risk:threat:type:taxonomy

A taxonomy of threat types. The `risk:threat:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:threat:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:tool:software

A software tool used in threat activity, as reported by a specific organization. The `risk:tool:software` type is derived from the base type: `guid`.

risk:tool:software:taxonomy

A taxonomy of software / tool types. The `risk:tool:software:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:tool:software:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:vuln

A unique vulnerability. The `risk:vuln` type is derived from the base type: `guid`.

risk:vuln:soft:range

A contiguous range of software versions which contain a vulnerability. The `risk:vuln:soft:range` type is derived from the base type: `guid`.

risk:vuln:type:taxonomy

A taxonomy of vulnerability types. The `risk:vuln:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `risk:vuln:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

risk:vulnerable

Indicates that a node is susceptible to a vulnerability. The `risk:vulnerable` type is derived from the base type: `guid`.

risk:vulnname

A vulnerability name such as log4j or rowhammer. The `risk:vulnname` type is derived from the base type: `str`.

The type `risk:vulnname` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `True`
- regex: `None`
- replace: `()`
- strip: `False`

rsa:key

An RSA keypair modulus and public exponent. The `rsa:key` type is derived from the base type: `comp`.

The type `rsa:key` has the following options set:

- fields: `(('mod', 'hex'), ('pub:exp', 'int'))`

sci:evidence

An assessment of how an observation supports or refutes a hypothesis. The `sci:evidence` type is derived from the base type: `guid`.

sci:experiment

An instance of running an experiment. The `sci:experiment` type is derived from the base type: `guid`.

sci:experiment:type:taxonomy

A taxonomy of experiment types. The `sci:experiment:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `sci:experiment:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

sci:hypothesis

A hypothesis or theory. The `sci:hypothesis` type is derived from the base type: `guid`.

sci:hypothesis:type:taxonomy

A taxonomy of hypothesis types. The `sci:hypothesis:type:taxonomy` type is derived from the base type: `taxonomy`.

The type `sci:hypothesis:type:taxonomy` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `False`

sci:observation

An observation which may have resulted from an experiment. The `sci:observation` type is derived from the base type: `guid`.

syn:cmd

A Synapse storm command. The `syn:cmd` type is derived from the base type: `str`.

The type `syn:cmd` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

syn:cron

A Cortex cron job. The `syn:cron` type is derived from the base type: `guid`.

syn:form

A Synapse form used for representing nodes in the graph. The `syn:form` type is derived from the base type: `str`.

The type `syn:form` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

syn:nodedata

A nodedata key and the form it may be present on. The `syn:nodedata` type is derived from the base type: `comp`.

The type `syn:nodedata` has the following options set:

- fields: `(('key', 'str'), ('form', 'syn:form'))`

syn:prop

A Synapse property. The `syn:prop` type is derived from the base type: `str`.

The type `syn:prop` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

syn:role

A Synapse role GUID. The `syn:role` type is derived from the base type: `guid`.

The type `syn:role` has the following options set:

- strip: `True`

syn:tagprop

A user defined tag property. The `syn:tagprop` type is derived from the base type: `str`.

The type `syn:tagprop` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`
- replace: `()`
- strip: `True`

syn:trigger

A Cortex trigger. The `syn:trigger` type is derived from the base type: `guid`.

syn:type

A Synapse type used for normalizing nodes and properties. The `syn:type` type is derived from the base type: `str`.

The type `syn:type` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `None`

- replace: ()
- strip: `True`

syn:user

A Synapse user GUID. The `syn:user` type is derived from the base type: `guid`.

The type `syn:user` has the following options set:

- strip: `True`

tel:call

A guid for a telephone call record. The `tel:call` type is derived from the base type: `guid`.

tel:mob:carrier

The fusion of a MCC/MNC. The `tel:mob:carrier` type is derived from the base type: `comp`.

The type `tel:mob:carrier` has the following options set:

- fields: `(('mcc', 'tel:mob:mcc'), ('mnc', 'tel:mob:mnc'))`

tel:mob:cell

A mobile cell site which a phone may connect to. The `tel:mob:cell` type is derived from the base type: `comp`.

The type `tel:mob:cell` has the following options set:

- fields: `(('carrier', 'tel:mob:carrier'), ('lac', ('int', {})), ('cid', ('int', {})))`

tel:mob:imid

Fused knowledge of an IMEI/IMSI used together. The `tel:mob:imid` type is derived from the base type: `comp`.

An example of `tel:mob:imid`:

- `(490154203237518, 310150123456789)`

The type `tel:mob:imid` has the following options set:

- fields: `(('imei', 'tel:mob:imei'), ('imsi', 'tel:mob:imsi'))`

tel:mob:imsiphone

Fused knowledge of an IMSI assigned phone number. The `tel:mob:imsiphone` type is derived from the base type: `comp`.

An example of `tel:mob:imsiphone`:

- `(310150123456789, "+7(495) 124-59-83")`

The type `tel:mob:imsiphone` has the following options set:

- fields: `(('imsi', 'tel:mob:imsi'), ('phone', 'tel:phone'))`

tel:mob:mcc

ITU Mobile Country Code. The `tel:mob:mcc` type is derived from the base type: `str`.

The type `tel:mob:mcc` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[0-9]{3}$`
- replace: `()`
- strip: `1`

tel:mob:mnc

ITU Mobile Network Code. The `tel:mob:mnc` type is derived from the base type: `str`.

The type `tel:mob:mnc` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `^[0-9]{2,3}$`
- replace: `()`
- strip: `1`

tel:mob:tac

A mobile Type Allocation Code. The `tel:mob:tac` type is derived from the base type: `int`.

An example of `tel:mob:tac`:

- `49015420`

The type `tel:mob:tac` has the following options set:

- enums:strict: `True`
- fmt: `%d`
- ismax: `False`
- ismin: `False`
- max: `None`
- min: `None`
- signed: `True`
- size: `8`

tel:mob:telem

A single mobile telemetry measurement. The `tel:mob:telem` type is derived from the base type: `guid`.

tel:txtmesg

A guid for an individual text message. The `tel:txtmesg` type is derived from the base type: `guid`.

transport:air:craft

An individual aircraft. The `transport:air:craft` type is derived from the base type: `guid`.

transport:air:flight

An individual instance of a flight. The `transport:air:flight` type is derived from the base type: `guid`.

transport:air:flightnum

A commercial flight designator including airline and serial. The `transport:air:flightnum` type is derived from the base type: `str`.

An example of `transport:air:flightnum`:

- `ua2437`

The type `transport:air:flightnum` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z]{2}[0-9]{1,4}$`
- replace: `((' ', ''),)`
- strip: `True`

transport:air:occupant

An occupant of a specific flight. The `transport:air:occupant` type is derived from the base type: `guid`.

transport:air:port

An IATA assigned airport code. The `transport:air:port` type is derived from the base type: `str`.

The type `transport:air:port` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `None`

- replace: ()
- strip: False

transport:air:tailnum

An aircraft registration number or military aircraft serial number. The `transport:air:tailnum` type is derived from the base type: `str`.

An example of `transport:air:tailnum`:

- `ff023`

The type `transport:air:tailnum` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^[a-z0-9-]{2,}$`
- replace: `()`
- strip: `True`

transport:air:telem

A telemetry sample from an aircraft in transit. The `transport:air:telem` type is derived from the base type: `guid`.

transport:direction

A direction measured in degrees with 0.0 being true North. The `transport:direction` type is derived from the base type: `hugenum`.

The type `transport:direction` has the following options set:

- modulo: `360`
- units: `None`

transport:land:license

A license to operate a land vehicle issued to a contact. The `transport:land:license` type is derived from the base type: `guid`.

transport:land:registration

Registration issued to a contact for a land vehicle. The `transport:land:registration` type is derived from the base type: `guid`.

transport:land:vehicle

An individual vehicle. The `transport:land:vehicle` type is derived from the base type: `guid`.

transport:sea:imo

An International Maritime Organization registration number. The `transport:sea:imo` type is derived from the base type: `str`.

The type `transport:sea:imo` has the following options set:

- globsuffix: `False`
- lower: `True`
- onespace: `False`
- regex: `^imo[0-9]{7}$`
- replace: `((' ', ''),)`
- strip: `True`

transport:sea:mmsi

A Maritime Mobile Service Identifier. The `transport:sea:mmsi` type is derived from the base type: `str`.

The type `transport:sea:mmsi` has the following options set:

- globsuffix: `False`
- lower: `False`
- onespace: `False`
- regex: `[0-9]{9}`
- replace: `()`
- strip: `False`

transport:sea:telem

A telemetry sample from a vessel in transit. The `transport:sea:telem` type is derived from the base type: `guid`.

transport:sea:vessel

An individual sea vessel. The `transport:sea:vessel` type is derived from the base type: `guid`.

12.2 Synapse Data Model - Forms

12.2.1 Forms

Forms are derived from types, or base types. Forms represent node types in the graph.

auth:access

An instance of using creds to access a resource.

The base type for the form can be found at *auth:access*.

Properties:

name	type	doc
:creds	*auth:creds*	The credentials used to attempt access.
:time	*time*	The time of the access attempt.
:success	*bool*	Set to true if the access was successful.
:person	*ps:person*	The person who attempted access.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 2 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

auth:creds

A unique set of credentials used to access a resource.

The base type for the form can be found at *auth:creds*.

Properties:

name	type	doc
:email	*inet:email*	The email address used to identify the user.
:user	*inet:user*	The user name used to identify the user.
:phone	*tel:phone*	The phone number used to identify the user.
:passwd	*inet:passwd*	The password used to authenticate.
:passwdhash	*it:auth:passwdhash*	The password hash used to authenticate.
:account	*it:account*	The account that the creds allow access to.
:website	*inet:url*	The base URL of the website that the credentials allow access to.
:host	*it:host*	The host that the credentials allow access to.
:wifi:ssid	*inet:wifi:ssid*	The WiFi SSID that the credentials allow access to.
:web:acct	*inet:web:acct*	The web account that the credentials allow access to.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

belief:subscriber

A contact which subscribes to a belief system.

The base type for the form can be found at *belief:subscriber*.

Properties:

name	type	doc
:contact	*ps:contact*	The contact which subscribes to the belief system.
:system	*belief:system*	The belief system to which the contact subscribes.
:began	*time*	The time that the contact began to be a subscriber to the belief system.
:ended	*time*	The time when the contact ceased to be a subscriber to the belief system.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirem	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
belief:subscr:	-(follows	belief:tene	The subscriber is assessed to generally adhere to the specific tenet.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

belief:system

A belief system such as an ideology, philosophy, or religion.

The base type for the form can be found at *belief:system*.

Properties:

name	type	doc	opts
:name	*str* onespace: True lower: True	The name of the belief system.	
:desc	*str*	A description of the belief system.	Display: {'hint': 'text'}
:type	*belief:system:type:taxono*	A taxonometric type for the belief system.	
:began	*time*	The time that the belief system was first observed.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
belief:syst	-(has)>	belief:tenet	The belief system includes the tenet.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 5 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

belief:system:type:taxonomy

A hierarchical taxonomy of belief system types.

The base type for the form can be found at *belief:system:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*belief:system:type:taxono*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

belief:tenet

A concrete tenet potentially shared by multiple belief systems.

The base type for the form can be found at *belief:tenet*.

Properties:

name	type	doc	opts
:name	*str* onespace: True lower: True	The name of the tenet.	
:desc	*str*	A description of the tenet.	Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
belief:subscriber	-(follows)>	belief:tenet	The subscriber is assessed to generally adhere to the spec
belief:system	-(has)>	belief:tenet	The belief system includes the tenet.
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target n
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target
it:exec:query	-(found)>	*	The target node was returned as a result of running the qu
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target nod
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on n

Table 7 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the co
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the ta
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:bundle

A bundle allows construction of products which bundle instances of other products.

The base type for the form can be found at *biz:bundle*.

Properties:

name	type	doc	opts
:count	*int*	The number of instances of the product or service included in the bundle.	
:price	*econ:price*	The price of the bundle.	
:product	*biz:product*	The product included in the bundle.	
:service	*biz:service*	The service included in the bundle.	
:deal	*biz:deal*	Deprecated. Please use econ:receipt:item for instances of bundles being sold.	Deprecated: True
:purchas	*econ:purchc*	Deprecated. Please use econ:receipt:item for instances of bundles being sold.	Deprecated: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 8 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:deal

A sales or procurement effort in pursuit of a purchase.

The base type for the form can be found at *biz:deal*.

Properties:

name	type	doc	opts
:title	*str*	A title for the deal.	
:type	*biz:dealtyp*	The type of deal.	Display: {'hint': 'taxonomy'}
:status	*biz:dealsta*	The status of the deal.	Display: {'hint': 'taxonomy'}
:updated	*time*	The last time the deal had a significant update.	
:contacted	*time*	The last time the contacts communicated about the deal.	
:rfp	*biz:rfp*	The RFP that the deal is in response to.	
:buyer	*ps:contact*	The primary contact information for the buyer.	
:buyer:org	*ou:org*	The buyer org.	
:buyer:orgn	*ou:name*	The reported ou:name of the buyer org.	
:buyer:orgf	*inet:fqdn*	The reported inet:fqdn of the buyer org.	
:seller	*ps:contact*	The primary contact information for the seller.	
:seller:org	*ou:org*	The seller org.	
:seller:org	*ou:name*	The reported ou:name of the seller org.	
:seller:org	*inet:fqdn*	The reported inet:fqdn of the seller org.	
:currency	*econ:curre*	The currency of econ:price values associated with the deal.	
:buyer:budg	*econ:price*	The buyers budget for the eventual purchase.	
:buyer:dead	*time*	When the buyer intends to make a decision.	
:offer:pric	*econ:price*	The total price of the offered products.	
:offer:expi	*time*	When the offer expires.	
:purchase	*econ:purch*	Records a purchase resulting from the deal.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 9 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:dealstatus

A deal/rfp status taxonomy.

The base type for the form can be found at *biz:dealstatus*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: `True` Display: `{'hint': 'text'}`
:desc	*str*	A definition of the taxonomy entry.	Display: `{'hint': 'text'}`
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: `True`
:depth	*int*	The depth indexed from 0.	Read Only: `True`
:parent	*biz:dealstatus*	The taxonomy parent.	Read Only: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 10 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:dealtype

A deal type taxonomy.

The base type for the form can be found at *biz:dealtype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*biz:dealtype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:listing

A product or service being listed for sale at a given price by a specific seller.

The base type for the form can be found at *biz:listing*.

Properties:

name	type	doc
:seller	*ps:contact*	The contact information for the seller.
:product	*biz:product*	The product being offered.
:service	*biz:service*	The service being offered.
:current	*bool*	Set to true if the offer is still current.
:time	*time*	The first known offering of this product/service by the organization for the asking price.
:expires	*time*	Set if the offer has a known expiration date.
:price	*econ:price*	The asking price of the product or service.
:currency	*econ:currenc*	The currency of the asking price.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:prodtype

A product type taxonomy.

The base type for the form can be found at *biz:prodtype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*biz:prodtype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 13 – continued from previous page

source	verb	target	doc
`ou:org`	`-(owns)>`	*	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

biz:product

A product which is available for purchase.

The base type for the form can be found at *biz:product*.

Properties:

name	type	doc	opts
:name	*str*	The name of the product.	
:type	*biz:prodtype*	The type of product.	Display: {'hint': 'taxonomy'}
:summary	*str*	A brief summary of the product.	Display: {'hint': 'text'}
:maker	*ps:contact*	A contact for the maker of the product.	
:madeby:org	*ou:org*	Deprecated. Please use biz:product:maker.	Deprecated: True
:madeby:orgname	*ou:name*	Deprecated. Please use biz:product:maker.	Deprecated: True
:madeby:orgfqdn	*inet:fqdn*	Deprecated. Please use biz:product:maker.	Deprecated: True
:price:retail	*econ:price*	The MSRP price of the product.	
:price:bottom	*econ:price*	The minimum offered or observed price of the product.	
:price:currency	*econ:currency*	The currency of the retail and bottom price properties.	
:bundles	*array* type: *biz:bundle* uniq: True sorted: True	An array of bundles included with the product.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 14 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:rfp

An RFP (Request for Proposal) soliciting proposals.

The base type for the form can be found at *biz:rfp*.

Properties:

name	type	doc	opts
:ext:id	*str*	An externally specified identifier for the RFP.	
:title	*str*	The title of the RFP.	
:summary	*str*	A brief summary of the RFP.	Display: {'hint': 'text'}
:status	*biz:dealstatus*	The status of the RFP.	Display: {'hint': 'enum'}
:url	*inet:url*	The official URL for the RFP.	
:file	*file:bytes*	The RFP document.	
:posted	*time*	The date/time that the RFP was posted.	
:quesdue	*time*	The date/time that questions are due.	
:propdue	*time*	The date/time that proposals are due.	
:contact	*ps:contact*	The contact information given for the org requesting offers.	
:purchases	*array* type: *econ:purchase* uniq: True sorted: True	Any known purchases that resulted from the RFP.	
:requirements	*array* type: *ou:goal* uniq: True sorted: True	A typed array which indexes each field.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 15 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:service

A service which is performed by a specific organization.

The base type for the form can be found at *biz:service*.

Properties:

name	type	doc	opts
:provider	*ps:contact*	The contact info of the entity which performs the service.	
:name	*str* lower: True onespace: True	The name of the service being performed.	
:summary	*str*	A brief summary of the service.	Display: {'hint': 'text'}
:type	*biz:service:type:taxono*	A taxonomy of service types.	
:launched	*time*	The time when the operator first made the service available.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 16 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

biz:stake

A stake or partial ownership in a company.

The base type for the form can be found at *biz:stake*.

Properties:

name	type	doc
:vitals	*ou:vitals*	The ou:vitals snapshot this stake is part of.
:org	*ou:org*	The resolved org.
:orgname	*ou:name*	The org name as reported by the source of the vitals.
:orgfqdn	*inet:fqdn*	The org FQDN as reported by the source of the vitals.
:name	*str*	An arbitrary name for this stake. Can be non-contact like “pool”.
:asof	*time*	The time the stake is being measured. Likely as part of an ou:vitals.
:shares	*int*	The number of shares represented by the stake.
:invested	*econ:price*	The amount of money invested in the cap table iteration.
:value	*econ:price*	The monetary value of the stake.
:percent	*hugenum*	The percentage ownership represented by this stake.
:owner	*ps:contact*	Contact information of the owner of the stake.
:purchase	*econ:purchase*	The purchase event for the stake.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:algorithm

A cryptographic algorithm name.

The base type for the form can be found at *crypto:algorithm*.

An example of `crypto:algorithm`:

- `aes256`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:currency:address

An individual crypto currency address.

The base type for the form can be found at *crypto:currency:address*.

An example of `crypto:currency:address`:

- `btc/1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2`

  Properties:

name	type	doc	opts
:coin	*crypto:currenc*	The crypto coin to which the address belongs.	Read Only: True
:seed	*crypto:key*	The cryptographic key and or password used to generate the address.	
:iden	*str*	The coin specific address identifier.	Read Only: True
:desc	*str*	A free-form description of the address.	
:contac	*ps:contact*	Contact information associated with the address.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next

Table 19 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:currency:block

An individual crypto currency block record on the blockchain.

The base type for the form can be found at *crypto:currency:block*.

Properties:

name	type	doc	opts
:coin	*crypto:currency:coin*	The coin/blockchain this block resides on.	Read Only: True
:offset	*int*	The index of this block.	Read Only: True
:hash	*hex*	The unique hash for the block.	
:minedby	*crypto:currency:addr*	The address which mined the block.	
:time	*time*	Time timestamp embedded in the block by the miner.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 20 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:currency:client

A fused node representing a crypto currency address used by an Internet client.

The base type for the form can be found at *crypto:currency:client*.

An example of `crypto:currency:client`:

- `(1.2.3.4, (btc, 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2))`

  Properties:

name	type	doc	opts
:inetac	*inet:client*	The Internet client address observed using the crypto currency address.	Read Only: True
:coinac	*crypto:currency:*	The crypto currency address observed in use by the Internet client.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:currency:coin

An individual crypto currency type.

The base type for the form can be found at *crypto:currency:coin*.

An example of `crypto:currency:coin`:

- `btc`

Properties:

name	type	doc
:name	*str*	The full name of the crypto coin.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:currency:transaction

An individual crypto currency transaction recorded on the blockchain.

The base type for the form can be found at *crypto:currency:transaction*.

Properties:

name	type	doc	opts
:hash	*hex*	The unique transaction hash for the transaction.	
:desc	*str*	An analyst specified description of the transaction.	
:block	*crypto:currency:block*	The block which records the transaction.	
:block:coin	*crypto:currency:coin*	The coin/blockchain of the block which records this transaction.	
:block:offset	*int*	The offset of the block which records this transaction.	
:success	*bool*	Set to true if the transaction was successfully executed and recorded.	
:status:code	*int*	A coin specific status code which may represent an error reason.	
:status:message	*str*	A coin specific status message which may contain an error reason.	
:to	*crypto:currency:addres*	The destination address of the transaction.	
:from	*crypto:currency:addres*	The source address of the transaction.	
:inputs	*array* type: *crypto:payment:input* sorted: True uniq: True	Deprecated. Please use crypto:payment:input:t	Deprecated: True
:outputs	*array* type: *crypto:payment:output* sorted: True uniq: True	Deprecated. Please use crypto:payment:output:	Deprecated: True
:fee	*econ:price*	The total fee paid to execute the transaction.	
:value	*econ:price*	The total value of the transaction.	
:time	*time*	The time this transaction was initiated.	
:eth:gasused	*int*	The amount of gas used to execute this transaction.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:key

A cryptographic key and algorithm.

The base type for the form can be found at *crypto:key*.

Properties:

name	type	doc	opts
:algorithm	*crypto:algorithm*	The cryptographic algorithm which uses the key material.	Example: aes256
:mode	*str* lower: True onespace: True	The algorithm specific mode in use.	
:iv	*hex*	The hex encoded initialization vector.	
:public	*hex*	The hex encoded public key material if the algorithm has a public/private key pair.	
:public:md5	*hash:md5*	The MD5 hash of the public key in raw binary form.	
:public:sha1	*hash:sha1*	The SHA1 hash of the public key in raw binary form.	
:public:sha256	*hash:sha256*	The SHA256 hash of the public key in raw binary form.	
:private	*hex*	The hex encoded private key material. All symmetric keys are private.	
:private:md5	*hash:md5*	The MD5 hash of the private key in raw binary form.	
:private:sha1	*hash:sha1*	The SHA1 hash of the private key in raw binary form.	
:private:sha256	*hash:sha256*	The SHA256 hash of the private key in raw binary form.	
:seed:passwd	*inet:passwd*	The seed password used to generate the key material.	
:seed:algorithm	*crypto:algorithm*	The algorithm used to generate the key from the seed password.	Example: pbkdf2

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:payment:input

A payment made into a transaction.

The base type for the form can be found at *crypto:payment:input*.

Properties:

name	type	doc
:transaction	*crypto:currency:transaction*	The transaction the payment was input to.
:address	*crypto:currency:address*	The address which paid into the transaction.
:value	*econ:price*	The value of the currency paid into the transaction.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 25 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:payment:output

A payment received from a transaction.

The base type for the form can be found at *crypto:payment:output*.

Properties:

name	type	doc
:transaction	*crypto:currency:transactic*	The transaction the payment was output from.
:address	*crypto:currency:address*	The address which received payment from the transaction.
:value	*econ:price*	The value of the currency received from the transaction.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 26 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:contract

A smart contract.

The base type for the form can be found at *crypto:smart:contract*.

Properties:

name	type	doc
:transaction	*crypto:currency:transaction*	The transaction which created the contract.
:address	*crypto:currency:address*	The address of the contract.
:bytecode	*file:bytes*	The bytecode which implements the contract.
:token:name	*str*	The ERC-20 token name.
:token:symbol	*str*	The ERC-20 token symbol.
:token:totalsupply	*hugenum*	The ERC-20 totalSupply value.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:burntoken

A smart contract effect which destroys a non-fungible token.

The base type for the form can be found at *crypto:smart:effect:burntoken*.

Properties:

name	type	doc
:token	*crypto:smart:token*	The non-fungible token that was destroyed.
:index	*int*	The order of the effect within the effects of one transaction.
:transactior	*crypto:currency:transacti*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:edittokensupply

A smart contract effect which increases or decreases the supply of a fungible token.

The base type for the form can be found at *crypto:smart:effect:edittokensupply*.

Properties:

name	type	doc
:contract	*crypto:smart:contract*	The contract which defines the tokens.
:amount	*hugenum*	The number of tokens added or removed if negative.
:totalsupply	*hugenum*	The total supply of tokens after this modification.
:index	*int*	The order of the effect within the effects of one transaction.
:transaction	*crypto:currency:transaction*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirement	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 29 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:minttoken

A smart contract effect which creates a new non-fungible token.

The base type for the form can be found at *crypto:smart:effect:minttoken*.

Properties:

name	type	doc
:token	*crypto:smart:token*	The non-fungible token that was created.
:index	*int*	The order of the effect within the effects of one transaction.
:transactior	*crypto:currency:transacti*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 30 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:proxytoken

A smart contract effect which grants a non-owner address the ability to manipulate a specific non-fungible token.

The base type for the form can be found at *crypto:smart:effect:proxytoken*.

Properties:

name	type	doc
:owner	*crypto:currency:addres*	The address granting proxy authority to manipulate non-fungible tokens.
:proxy	*crypto:currency:addres*	The address granted proxy authority to manipulate non-fungible tokens.
:token	*crypto:smart:token*	The specific token being granted access to.
:index	*int*	The order of the effect within the effects of one transaction.
:transacti(	*crypto:currency:transa*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:proxytokenall

A smart contract effect which grants a non-owner address the ability to manipulate all non-fungible tokens of the owner.

The base type for the form can be found at *crypto:smart:effect:proxytokenall*.

Properties:

name	type	doc
:contract	*crypto:smart:contra*	The contract which defines the tokens.
:owner	*crypto:currency:ad*	The address granting/denying proxy authority to manipulate all non-fungible tokens of the owner.
:proxy	*crypto:currency:ad*	The address granted/denied proxy authority to manipulate all non-fungible tokens of the owner.
:approval	*bool*	The approval status.
:index	*int*	The order of the effect within the effects of one transaction.
:transact	*crypto:currency:tra*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:proxytokens

A smart contract effect which grants a non-owner address the ability to manipulate fungible tokens.

The base type for the form can be found at *crypto:smart:effect:proxytokens*.

Properties:

name	type	doc
:contract	*crypto:smart:contract*	The contract which defines the tokens.
:owner	*crypto:currency:addres.*	The address granting proxy authority to manipulate fungible tokens.
:proxy	*crypto:currency:addres.*	The address granted proxy authority to manipulate fungible tokens.
:amount	*hex*	The hex encoded amount of tokens the proxy is allowed to manipulate.
:index	*int*	The order of the effect within the effects of one transaction.
:transactic	*crypto:currency:transac*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 33 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:transfertoken

A smart contract effect which transfers ownership of a non-fungible token.

The base type for the form can be found at *crypto:smart:effect:transfertoken*.

Properties:

name	type	doc
:token	*crypto:smart:token*	The non-fungible token that was transferred.
:from	*crypto:currency:address*	The address the NFT was transferred from.
:to	*crypto:currency:address*	The address the NFT was transferred to.
:index	*int*	The order of the effect within the effects of one transaction.
:transactior	*crypto:currency:transacti*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 34 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:effect:transfertokens

A smart contract effect which transfers fungible tokens.

The base type for the form can be found at *crypto:smart:effect:transfertokens*.

Properties:

name	type	doc
:contract	*crypto:smart:contract*	The contract which defines the tokens.
:from	*crypto:currency:address*	The address the tokens were transferred from.
:to	*crypto:currency:address*	The address the tokens were transferred to.
:amount	*hugenum*	The number of tokens transferred.
:index	*int*	The order of the effect within the effects of one transaction.
:transactior	*crypto:currency:transacti*	The transaction where the smart contract was called.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:smart:token

A token managed by a smart contract.

The base type for the form can be found at *crypto:smart:token*.

Properties:

name	type	doc	opts
:contract	*crypto:smart:con*	The smart contract which defines and manages the token.	Read Only: True
:tokenid	*hugenum*	The token ID.	Read Only: True
:owner	*crypto:currency:*	The address which currently owns the token.	
:nft:url	*inet:url*	The URL which hosts the NFT metadata.	
:nft:meta	*data*	The raw NFT metadata.	
:nft:meta:name	*str*	The name field from the NFT metadata.	
:nft:meta:desc:	*str*	The description field from the NFT metadata.	Display: {'hint': 'text'}
:nft:meta:imag	*inet:url*	The image URL from the NFT metadata.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 36 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:x509:cert

A unique X.509 certificate.

The base type for the form can be found at *crypto:x509:cert*.

Properties:

name	type	doc
:file	*file:bytes*	The file that the certificate metadata was parsed from.
:subject	*str*	The subject identifier, commonly in X.500/LDAP format, to which the certificate was issued.
:issuer	*str*	The Distinguished Name (DN) of the Certificate Authority (CA) which issued the certificate.
:issuer:cert	*crypto:x509:cert*	The certificate used by the issuer to sign this certificate.
:serial	*hex* zeropad: `40`	The certificate serial number as a big endian hex value.
:version	*int* enums: `((0, 'v1'), (2, 'v3'))`	The version integer in the certificate. (ex. 2 == v3).
:validity:notbefore	*time*	The timestamp for the beginning of the certificate validity period.
:validity:notafter	*time*	The timestamp for the end of the certificate validity period.
:md5	*hash:md5*	The MD5 fingerprint for the certificate.
:sha1	*hash:sha1*	The SHA1 fingerprint for the certificate.
:sha256	*hash:sha256*	The SHA256 fingerprint for the certificate.
:rsa:key	*rsa:key*	The optional RSA public key associated with the certificate.
:algo	*iso:oid*	The X.509 signature algorithm OID.
:signature	*hex*	The hexadecimal representation of the digital signature.
:ext:sans	*array* type: *crypto:x509:san* uniq: `True` sorted: `True`	The Subject Alternate Names (SANs) listed in the certificate.
:ext:crls	*array* type: *crypto:x509:san* uniq: `True` sorted: `True`	A list of Subject Alternate Names (SANs) for Distribution Points.
:identities:fqdns	*array* type: *inet:fqdn* uniq: `True`	The fused list of FQDNs identified by the cert CN and SANs.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:x509:crl

A unique X.509 Certificate Revocation List.

The base type for the form can be found at *crypto:x509:crl*.

Properties:

name	type	doc
:file	*file:bytes*	The file containing the CRL.
:url	*inet:url*	The URL where the CRL was published.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 38 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:x509:revoked

A revocation relationship between a CRL and an X.509 certificate.

The base type for the form can be found at *crypto:x509:revoked*.

Properties:

name	type	doc	opts
:crl	*crypto:x509:crl*	The CRL which revoked the certificate.	Read Only: True
:cert	*crypto:x509:cert*	The certificate revoked by the CRL.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 39 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

crypto:x509:signedfile

A digital signature relationship between an X.509 certificate and a file.

The base type for the form can be found at *crypto:x509:signedfile*.

Properties:

name	type	doc	opts
:cert	*crypto:x509:cert*	The certificate for the key which signed the file.	Read Only: True
:file	*file:bytes*	The file which was signed by the certificates key.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 40 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:acct:balance

A snapshot of the balance of an account at a point in time.

The base type for the form can be found at *econ:acct:balance*.

Properties:

name	type	doc
:time	*time*	The time the balance was recorded.
:pay:card	*econ:pay:card*	The payment card holding the balance.
:crypto:address	*crypto:currency:address*	The crypto currency address holding the balance.
:amount	*econ:price*	The account balance at the time.
:currency	*econ:currency*	The currency of the balance amount.
:delta	*econ:price*	The change since last regular sample.
:total:received	*econ:price*	The total amount of currency received by the account.
:total:sent	*econ:price*	The total amount of currency sent from the account.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:acct:invoice

An invoice issued requesting payment.

The base type for the form can be found at *econ:acct:invoice*.

Properties:

name	type	doc
:issued	*time*	The time that the invoice was issued to the recipient.
:issuer	*ps:contact*	The contact information for the entity who issued the invoice.
:purchase	*econ:purchase*	The purchase that the invoice is requesting payment for.
:recipient	*ps:contact*	The contact information for the intended recipient of the invoice.
:due	*time*	The time by which the payment is due.
:paid	*bool*	Set to true if the invoice has been paid in full.
:amount	*econ:price*	The balance due.
:currency	*econ:currency*	The currency that the invoice specifies for payment.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 42 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:acct:payment

A payment or crypto currency transaction.

The base type for the form can be found at *econ:acct:payment*.

Properties:

name	type	doc
:txnid	*str* strip: True	A payment processor specific transaction id.
:fee	*econ:price*	The transaction fee paid by the recipient to the payment processor.
:from:account	*econ:bank:account*	The bank account which made the payment.
:from:pay:card	*econ:pay:card*	The payment card making the payment.
:from:contract	*ou:contract*	A contract used as an aggregate payment source.
:from:coinaddr	*crypto:currency:address*	The crypto currency address making the payment.
:from:contact	*ps:contact*	Contact information for the entity making the payment.
:to:account	*econ:bank:account*	The bank account which received the payment.
:to:coinaddr	*crypto:currency:address*	The crypto currency address receiving the payment.
:to:contact	*ps:contact*	Contact information for the person/org being paid.
:to:contract	*ou:contract*	A contract used as an aggregate payment destination.
:time	*time*	The time the payment was processed.
:purchase	*econ:purchase*	The purchase which the payment was paying for.
:amount	*econ:price*	The amount of money transferred in the payment.
:currency	*econ:currency*	The currency of the payment.
:memo	*str*	A small note specified by the payer common in financial transactions.
:crypto:transaction	*crypto:currency:transaction*	A crypto currency transaction that initiated the payment.
:invoice	*econ:acct:invoice*	The invoice that the payment applies to.
:receipt	*econ:acct:receipt*	The receipt that was issued for the payment.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:bank:statement	-(has)>	econ:acct:payment	The bank statement includes the payment.
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting th
it:exec:query	-(found)>	*	The target node was returned as a result of runnin
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of t
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the ta
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target n
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target n
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the at
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result o
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access
risk:leak	-(leaked)>	*	The leak included the disclosure of the target nod
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the targ
sci:experiment	-(uses)>	*	The experiment used the target nodes when it wa
sci:observation	-(has)>	*	The observations are summarized from the target

econ:acct:receipt

A receipt issued as proof of payment.

The base type for the form can be found at *econ:acct:receipt*.

Properties:

name	type	doc
:issued	*time*	The time the receipt was issued.
:purchase	*econ:purchase*	The purchase that the receipt confirms payment for.
:issuer	*ps:contact*	The contact information for the entity who issued the receipt.
:recipient	*ps:contact*	The contact information for the entity who received the receipt.
:currency	*econ:currency*	The currency that the receipt uses to specify the price.
:amount	*econ:price*	The price that the receipt confirms was paid.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:acquired

Deprecated. Please use econ:purchase -(acquired)> *.

The base type for the form can be found at *econ:acquired*.

Properties:

name	type	doc	opts
:purchase	*econ:purchase*	The purchase event which acquired an item.	Read Only: True
:item	*ndef*	A reference to the item that was acquired.	Read Only: True
:item:form	*str*	The form of item purchased.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 45 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:aba:rtn

An American Bank Association (ABA) routing transit number (RTN).

The base type for the form can be found at *econ:bank:aba:rtn*.

Properties:

name	type	doc
:bank	*ou:org*	The bank which was issued the ABA RTN.
:bank:name	*ou:name*	The name which is registered for this ABA RTN.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.

continues on next page

Table 46 – continued from previous page

source	verb	target	doc
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:account

A bank account.

The base type for the form can be found at *econ:bank:account*.

Properties:

name	type	doc
:type	*econ:bank:account:type:taxono*	The type of bank account.
:aba:rtn	*econ:bank:aba:rtn*	The ABA routing transit number for the bank which issued the account.
:number	*str* regex: [0-9]+	The account number.
:iban	*econ:bank:iban*	The IBAN for the account.
:contact	*ps:contact*	The contact information associated with the bank account.
:issuer	*ou:org*	The bank which issued the account.
:issuer:name	*ou:name*	The name of the bank which issued the account.
:currency	*econ:currency*	The currency of the account balance.
:balance	*econ:bank:balance*	The most recently known bank balance information.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:account:type:taxonomy

A bank account type taxonomy.

The base type for the form can be found at *econ:bank:account:type:taxonomy*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:balance

A balance contained by a bank account at a point in time.

The base type for the form can be found at *econ:bank:balance*.

Properties:

name	type	doc
:time	*time*	The time that the account balance was observed.
:amount	*econ:price*	The amount of currency available at the time.
:account	*econ:bank:account*	The bank account which contained the balance amount.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 49 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:iban

An International Bank Account Number.

The base type for the form can be found at *econ:bank:iban*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 50 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:statement

A statement of bank account payment activity over a period of time.

The base type for the form can be found at *econ:bank:statement*.

Properties:

name	type	doc
:account	*econ:bank:accoun*	The bank account used to compute the statement.
:period	*ival*	The period that the statement includes.
:starting:balanc	*econ:price*	The account balance at the beginning of the statement period.
:ending:balance	*econ:price*	The account balance at the end of the statement period.

Source Edges:

source	verb	target	doc
*	-(meets):	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat	geo:telem	The source node was seen at the geo:telem node place and time.
econ:bank:stat€	-(has)>	econ:acct:pay	The bank statement includes the payment.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next page

Table 51 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:bank:swift:bic

A Society for Worldwide Interbank Financial Telecommunication (SWIFT) Business Identifier Code (BIC).

The base type for the form can be found at *econ:bank:swift:bic*.

Properties:

name	type	doc
:business	*ou:org*	The business which is the registered owner of the SWIFT BIC.
:office	*ps:contact*	The branch or office which is specified in the last 3 digits of the SWIFT BIC.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:fin:bar

A sample of the open, close, high, low prices of a security in a specific time window.

The base type for the form can be found at *econ:fin:bar*.

Properties:

name	type	doc
:security	*econ:fin:security*	The security measured by the bar.
:ival	*ival*	The interval of measurement.
:price:open	*econ:price*	The opening price of the security.
:price:close	*econ:price*	The closing price of the security.
:price:low	*econ:price*	The low price of the security.
:price:high	*econ:price*	The high price of the security.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:fin:exchange

A financial exchange where securities are traded.

The base type for the form can be found at *econ:fin:exchange*.

Properties:

name	type	doc	opts
:name	*str* lower: True strip: True	A simple name for the exchange.	Example: nasdaq
:org	*ou:org*	The organization that operates the exchange.	
:currency	*econ:currency*	The currency used for all transactions in the exchange.	Example: usd

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.

continues on next page

Table 54 – continued from previous page

source	verb	target	doc
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:fin:security

A financial security which is typically traded on an exchange.

The base type for the form can be found at *econ:fin:security*.

Properties:

name	type	doc
:exchange	*econ:fin:exchange*	The exchange on which the security is traded.
:ticker	*str* lower: True strip: True	The identifier for this security within the exchange.
:type	*str* lower: True strip: True	A user defined type such as stock, bond, option, future, or forex.
:price	*econ:price*	The last known/available price of the security.
:time	*time*	The time of the last know price sample.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:fin:tick

A sample of the price of a security at a single moment in time.

The base type for the form can be found at *econ:fin:tick*.

Properties:

name	type	doc
:security	*econ:fin:security*	The security measured by the tick.
:time	*time*	The time the price was sampled.
:price	*econ:price*	The price of the security at the time.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 56 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:pay:card

A single payment card.

The base type for the form can be found at *econ:pay:card*.

Properties:

name	type	doc
:pan	*econ:pay:pan*	The payment card number.
:pan:mii	*econ:pay:mii*	The payment card MII.
:pan:iin	*econ:pay:iin*	The payment card IIN.
:name	*ps:name*	The name as it appears on the card.
:expr	*time*	The expiration date for the card.
:cvv	*econ:pay:cvv*	The Card Verification Value on the card.
:pin	*econ:pay:pin*	The Personal Identification Number on the card.
:account	*econ:bank:account*	A bank account associated with the payment card.
:contact	*ps:contact*	The contact information associated with the payment card.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 57 – continued from previous page

source	verb	target	doc
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:pay:iin

An Issuer Id Number (IIN).

The base type for the form can be found at *econ:pay:iin*.

Properties:

name	type	doc
:org	*ou:org*	The issuer organization.
:name	*str* lower: True	The registered name of the issuer.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:purchase

A purchase event.

The base type for the form can be found at *econ:purchase*.

Properties:

name	type	doc
:by:contact	*ps:contact*	The contact information used to make the purchase.
:from:contact	*ps:contact*	The contact information used to sell the item.
:time	*time*	The time of the purchase.
:place	*geo:place*	The place where the purchase took place.
:paid	*bool*	Set to True if the purchase has been paid in full.
:paid:time	*time*	The point in time where the purchase was paid in full.
:settled	*time*	The point in time where the purchase was settled.
:campaign	*ou:campaign*	The campaign that the purchase was in support of.
:price	*econ:price*	The econ:price of the purchase.
:currency	*econ:currency*	The econ:price of the purchase.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
econ:purcha	-(acquired)	*	The purchase was used to acquire the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

econ:receipt:item

A line item included as part of a purchase.

The base type for the form can be found at *econ:receipt:item*.

Properties:

name	type	doc
:purchase	*econ:purchase*	The purchase that contains this line item.
:count	*int* min: 1	The number of items included in this line item.
:price	*econ:price*	The total cost of this receipt line item.
:product	*biz:product*	The product being being purchased in this line item.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 60 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

edge:has

A digraph edge which records that N1 has N2.

The base type for the form can be found at *edge:has*.

Properties:

name	type	doc	opts
:n1	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n1:form	*str*	The base string type.	Read Only: True
:n2	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n2:form	*str*	The base string type.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 61 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

edge:refs

A digraph edge which records that N1 refers to or contains N2.

The base type for the form can be found at *edge:refs*.

Properties:

name	type	doc	opts
:n1	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n1:form	*str*	The base string type.	Read Only: True
:n2	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n2:form	*str*	The base string type.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

edge:wentto

A digraph edge which records that N1 went to N2 at a specific time.

The base type for the form can be found at *edge:wentto*.

Properties:

name	type	doc	opts
:n1	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n1:form	*str*	The base string type.	Read Only: True
:n2	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n2:form	*str*	The base string type.	Read Only: True
:time	*time*	A date/time value.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 63 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

edu:class

An instance of an edu:course taught at a given time.

The base type for the form can be found at *edu:class*.

Properties:

name	type	doc
:course	*edu:course*	The course being taught in the class.
:instructor	*ps:contact*	The primary instructor for the class.
:assistants	*array* type: *ps:contact* uniq: True sorted: True	An array of assistant/co-instructor contacts.
:date:first	*time*	The date of the first day of class.
:date:last	*time*	The date of the last day of class.
:isvirtual	*bool*	Set if the class is known to be virtual.
:virtual:url	*inet:url*	The URL a student would use to attend the virtual class.
:virtual:provider	*ps:contact*	Contact info for the virtual infrastructure provider.
:place	*geo:place*	The place that the class is held.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

edu:course

A course of study taught by an org.

The base type for the form can be found at *edu:course*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name of the course.	Example: organic chemistry for beginners
:desc	*str*	A brief course description.	
:code	*str* lower: True strip: True	The course catalog number or designator.	Example: chem101
:institution	*ps:contact*	The org or department which teaches the course.	
:prereqs	*array* type: *edu:course* uniq: True sorted: True	The pre-requisite courses for taking this course.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 65 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:archive:entry

An archive entry representing a file and metadata within a parent archive file.

The base type for the form can be found at *file:archive:entry*.

Properties:

name	type	doc
:parent	*file:bytes*	The parent archive file.
:file	*file:bytes*	The file contained within the archive.
:path	*file:path*	The file path of the archived file.
:user	*inet:user*	The name of the user who owns the archived file.
:added	*time*	The time that the file was added to the archive.
:created	*time*	The created time of the archived file.
:modified	*time*	The modified time of the archived file.
:comment	*str*	The comment field for the file entry within the archive.
:posix:uid	*int*	The POSIX UID of the user who owns the archived file.
:posix:gid	*int*	The POSIX GID of the group who owns the archived file.
:posix:perms	*int*	The POSIX permissions mask of the archived file.
:archived:size	*int*	The encoded or compressed size of the archived file within the parent.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:base

A file name with no path.

The base type for the form can be found at *file:base*.

An example of `file:base`:

- `woot.exe`

Properties:

name	type	doc	opts
`:ext`	*str*	The file extension (if any).	Read Only: `True`

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	`*`	The meta:rule is designed to detect instances of the target nod
`meta:rule`	`-(matches)>`	`*`	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	`*`	The meta:source observed the target node.
`ou:campaign`	`-(targets)>`	`*`	The campaign targeted the target nodes.
`ou:campaign`	`-(uses)>`	`*`	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	`*`	The contribution includes the specific node.
`ou:org`	`-(has)>`	`*`	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	`*`	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	`*`	The organization targets the target node.
`ou:org`	`-(uses)>`	`*`	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	`*`	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	`*`	The contact owns or owned the target node.
`ps:person`	`-(has)>`	`*`	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	`*`	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	`*`	The attack targeted the target node.
`risk:attack`	`-(uses)>`	`*`	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	`*`	The target node was stolen or copied as a result of the compro
`risk:extortion`	`-(leveraged)>`	`*`	The extortion event was based on attacker access to the target

continues on next

Table 67 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:bytes

The file bytes type with SHA256 based primary property.

The base type for the form can be found at *file:bytes*.

Properties:

name	type	doc
:size	*int*	The file size in bytes.
:md5	*hash:md5*	The md5 hash of the file.
:sha1	*hash:sha1*	The sha1 hash of the file.
:sha256	*hash:sha25*	The sha256 hash of the file.
:sha512	*hash:sha51*	The sha512 hash of the file.
:name	*file:base*	The best known base name for the file.
:mime	*file:mime*	The "best" mime type name for the file.
:mime:x509:cn	*str*	The Common Name (CN) attribute of the x509 Subject.
:mime:pe:size	*int*	The size of the executable file according to the PE file header.
:mime:pe:imphash	*hash:md5*	The PE import hash of the file as calculated by pefile; https://github.com/erocarrera/pefile .
:mime:pe:compiled	*time*	The compile time of the file according to the PE header.
:mime:pe:pdbpath	*file:path*	The PDB string according to the PE.
:mime:pe:exports:	*time*	The export time of the file according to the PE.
:mime:pe:exports:	*str*	The export library name according to the PE.
:mime:pe:richhdr	*hash:sha25*	The sha256 hash of the rich header bytes.
:exe:compiler	*it:prod:sofi*	The software used to compile the file.
:exe:packer	*it:prod:sofi*	The packer software used to encode the file.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:filepath

The fused knowledge of the association of a file:bytes node and a file:path.

The base type for the form can be found at *file:filepath*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file seen at a path.	Read Only: True
:path	*file:path*	The path a file was seen at.	Read Only: True
:path:dir	*file:path*	The parent directory.	Read Only: True
:path:base	*file:base*	The name of the file.	Read Only: True
:path:base:ext	*str*	The extension of the file name.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:ismime

Records one, of potentially multiple, mime types for a given file.

The base type for the form can be found at *file:ismime*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file node that is an instance of the named mime type.	Read Only: True
:mime	*file:mime*	The mime type of the file.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 70 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime

A file mime name string.

The base type for the form can be found at *file:mime*.

An example of `file:mime`:

- `text/plain`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next

Table 71 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:gif

The GUID of a set of mime metadata for a .gif file.

The base type for the form can be found at *file:mime:gif* .

Properties:

name	type	doc
:desc	*str*	MIME specific description field extracted from metadata.
:comment	*str*	MIME specific comment field extracted from metadata.
:created	*time*	MIME specific creation timestamp extracted from metadata.
:imageid	*str*	MIME specific unique identifier extracted from metadata.
:author	*ps:contact*	MIME specific contact information extracted from metadata.
:latlong	*geo:latlong*	MIME specific lat/long information extracted from metadata.
:altitude	*geo:altitude*	MIME specific altitude information extracted from metadata.
:text	*str* lower: True onespace: True	The text contained within the image.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:jpg

The GUID of a set of mime metadata for a .jpg file.

The base type for the form can be found at *file:mime:jpg*.

Properties:

name	type	doc
:desc	*str*	MIME specific description field extracted from metadata.
:comment	*str*	MIME specific comment field extracted from metadata.
:created	*time*	MIME specific creation timestamp extracted from metadata.
:imageid	*str*	MIME specific unique identifier extracted from metadata.
:author	*ps:contact*	MIME specific contact information extracted from metadata.
:latlong	*geo:latlong*	MIME specific lat/long information extracted from metadata.
:altitude	*geo:altitude*	MIME specific altitude information extracted from metadata.
:text	*str* lower: `True` onespace: `True`	The text contained within the image.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:macho:loadcmd

A generic load command pulled from the Mach-O headers.

The base type for the form can be found at *file:mime:macho:loadcmd*.

Properties:

name	type	doc
:file	*file:bytes*	The Mach-O file containing the load command.
:type	*int* enums: ((1, 'segment'), (2, 'symbol table'), (3, 'gdb symbol table'), (4, 'thread'), (5, 'unix thread'), (6, 'fixed VM shared library'), (7, 'fixed VM shared library identification'), (8, 'object identification'), (9, 'fixed VM file inclusion'), (10, 'prepage'), (11, 'dynamic link-edit symbol table'), (12, 'load dynamically linked shared library'), (13, 'dynamically linked shared library identifier'), (14, 'load dynamic linker'), (15, 'dynamic linker identification'), (16, 'prebound dynamically linked shared library'), (17, 'image routines'), (18, 'sub framework'), (19, 'sub umbrella'), (20, 'sub client'), (21, 'sub library'), (22, 'two level namespace lookup hints'), (23, 'prebind checksum'), (24, 'weak import dynamically linked shared library'), (25, '64bit segment'), (26, '64bit image routines'), (27, 'uuid'), (28, 'runpath additions'), (29, 'code signature'), (30, 'split segment info'), (31, 'load and re-export dynamic library'), (32, 'delay load of dynamic library'), (33, 'encrypted segment	The type of the load command.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:macho:section

A section inside a Mach-O binary denoting a named region of bytes inside a segment.

The base type for the form can be found at *file:mime:macho:section*.

Properties:

name	type	doc
:segment	*file:mime:macho:segment*	The Mach-O segment that contains this section.
:name	*str*	Name of the section.
:size	*int*	Size of the section in bytes.
:type	*int* enums: ((0, 'regular'), (1, 'zero fill on demand'), (2, 'only literal C strings'), (3, 'only 4 byte literals'), (4, 'only 8 byte literals'), (5, 'only pointers to literals'), (6, 'only non-lazy symbol pointers'), (7, 'only lazy symbol pointers'), (8, 'only symbol stubs'), (9, 'only function pointers for init'), (10, 'only function pointers for fini'), (11, 'contains symbols to be coalesced'), (12, 'zero fill on deman (greater than 4gb)'), (13, 'only pairs of function pointers for interposing'), (14, 'only 16 byte literals'), (15, 'dtrace object format'), (16, 'only lazy symbols pointers to lazy dynamic libraries'))	The type of the section.
:sha256	*hash:sha256*	The sha256 hash of the bytes of the Mach-O section.
:offset	*int*	The file offset to the beginning of the section.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:macho:segment

A named region of bytes inside a Mach-O binary.

The base type for the form can be found at *file:mime:macho:segment*.

Properties:

name	type	doc
:name	*str*	The name of the Mach-O segment.
:memsize	*int*	The size of the segment in bytes, when resident in memory, according to the load command structure.
:disksize	*int*	The size of the segment in bytes, when on disk, according to the load command structure.
:sha256	*hash:sha256*	The sha256 hash of the bytes of the segment.
:offset	*int*	The file offset to the beginning of the segment.
:file	*file:bytes*	The Mach-O file containing the load command.
:type	*int* enums: ((1, 'segment'), (2, 'symbol table'), (3, 'gdb symbol table'), (4, 'thread'), (5, 'unix thread'), (6, 'fixed VM shared library'), (7, 'fixed VM shared library identification'), (8, 'object identification'), (9, 'fixed VM file inclusion'), (10, 'prepage'), (11, 'dynamic link-edit symbol table'), (12, 'load dynamically linked shared library'), (13, 'dynamically linked shared library identifier'), (14, 'load dynamic linker'), (15, 'dynamic linker identification'), (16, 'prebound dynamically linked shared library'), (17, 'image routines'), (18, 'sub framework'), (19, 'sub umbrella'), (20, 'sub client'), (21, 'sub library'), (22, 'two level namespace lookup hints'), (23, 'prebind checksum'), (24, 'weak import dynamically	The type of the load command.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:macho:uuid

A specific load command denoting a UUID used to uniquely identify the Mach-O binary.

The base type for the form can be found at *file:mime:macho:uuid*.

Properties:

name	type	doc
:uuid	*guid*	The UUID of the Mach-O application (as defined in an LC_UUID load command).
:file	*file:bytes*	The Mach-O file containing the load command.
:type	*int* enums: ((1, 'segment'), (2, 'symbol table'), (3, 'gdb symbol table'), (4, 'thread'), (5, 'unix thread'), (6, 'fixed VM shared library'), (7, 'fixed VM shared library identification'), (8, 'object identification'), (9, 'fixed VM file inclusion'), (10, 'prepage'), (11, 'dynamic link-edit symbol table'), (12, 'load dynamically linked shared library'), (13, 'dynamically linked shared library identifier'), (14, 'load dynamic linker'), (15, 'dynamic linker identification'), (16, 'prebound dynamically linked shared library'), (17, 'image routines'), (18, 'sub framework'), (19, 'sub umbrella'), (20, 'sub client'), (21, 'sub library'), (22, 'two level namespace lookup hints'), (23, 'prebind checksum'), (24, 'weak import dynamically linked shared library'), (25, '64bit segment'), (26, '64bit image routines'), (27, 'uuid'), (28, 'runpath additions'), (29, 'code signature'), (30, 'split segment info'), (31, 'load and re-export dynamic library'), (32, 'delay	The type of the load command.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:macho:version

A specific load command used to denote the version of the source used to build the Mach-O binary.

The base type for the form can be found at *file:mime:macho:version*.

Properties:

name	type	doc
:version	*str*	The version of the Mach-O file encoded in an LC_VERSION load command.
:file	*file:bytes*	The Mach-O file containing the load command.
:type	*int* enums: ((1, 'segment'), (2, 'symbol table'), (3, 'gdb symbol table'), (4, 'thread'), (5, 'unix thread'), (6, 'fixed VM shared library'), (7, 'fixed VM shared library identification'), (8, 'object identification'), (9, 'fixed VM file inclusion'), (10, 'prepage'), (11, 'dynamic link-edit symbol table'), (12, 'load dynamically linked shared library'), (13, 'dynamically linked shared library identifier'), (14, 'load dynamic linker'), (15, 'dynamic linker identification'), (16, 'prebound dynamically linked shared library'), (17, 'image routines'), (18, 'sub framework'), (19, 'sub umbrella'), (20, 'sub client'), (21, 'sub library'), (22, 'two level namespace lookup hints'), (23, 'prebind checksum'), (24, 'weak import dynamically linked shared library'), (25, '64bit segment'), (26, '64bit image routines'), (27, 'uuid'), (28, 'runpath additions'), (29, 'code signature'), (30, 'split segment info'), (31, 'load and re-export dynamic library'), (32, 'delay	The type of the load command.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:msdoc

The GUID of a set of mime metadata for a Microsoft Word file.

The base type for the form can be found at *file:mime:msdoc*.

Properties:

name	type	doc
:title	*str*	The title extracted from Microsoft Office metadata.
:author	*str*	The author extracted from Microsoft Office metadata.
:subject	*str*	The subject extracted from Microsoft Office metadata.
:application	*str*	The creating_application extracted from Microsoft Office metadata.
:created	*time*	The create_time extracted from Microsoft Office metadata.
:lastsaved	*time*	The last_saved_time extracted from Microsoft Office metadata.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 79 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:msppt

The GUID of a set of mime metadata for a Microsoft Powerpoint file.

The base type for the form can be found at *file:mime:msppt*.

Properties:

name	type	doc
:title	*str*	The title extracted from Microsoft Office metadata.
:author	*str*	The author extracted from Microsoft Office metadata.
:subject	*str*	The subject extracted from Microsoft Office metadata.
:application	*str*	The creating_application extracted from Microsoft Office metadata.
:created	*time*	The create_time extracted from Microsoft Office metadata.
:lastsaved	*time*	The last_saved_time extracted from Microsoft Office metadata.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 80 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:msxls

The GUID of a set of mime metadata for a Microsoft Excel file.

The base type for the form can be found at *file:mime:msxls*.

Properties:

name	type	doc
:title	*str*	The title extracted from Microsoft Office metadata.
:author	*str*	The author extracted from Microsoft Office metadata.
:subject	*str*	The subject extracted from Microsoft Office metadata.
:application	*str*	The creating_application extracted from Microsoft Office metadata.
:created	*time*	The create_time extracted from Microsoft Office metadata.
:lastsaved	*time*	The last_saved_time extracted from Microsoft Office metadata.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:pe:export

The fused knowledge of a file:bytes node containing a pe named export.

The base type for the form can be found at *file:mime:pe:export*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file containing the export.	Read Only: True
:name	*str*	The name of the export in the file.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 82 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:pe:resource

The fused knowledge of a file:bytes node containing a pe resource.

The base type for the form can be found at *file:mime:pe:resource*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file containing the resource.	Read Only: True
:type	*pe:resource:type*	The typecode for the resource.	Read Only: True
:langid	*pe:langid*	The language code for the resource.	Read Only: True
:resource	*file:bytes*	The sha256 hash of the resource bytes.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 83 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:pe:section

The fused knowledge a file:bytes node containing a pe section.

The base type for the form can be found at *file:mime:pe:section*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file containing the section.	Read Only: True
:name	*str*	The textual name of the section.	Read Only: True
:sha25(	*hash:sha25*	The sha256 hash of the section. Relocations must be zeroed before hashing.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 84 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:pe:vsvers:info

knowledge of a file:bytes node containing vsvers info.

The base type for the form can be found at *file:mime:pe:vsvers:info*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file containing the vsversion keyval pair.	Read Only: True
:keyval	*file:mime:pe:vsvers:keyv*	The vsversion info keyval in this file:bytes node.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:pe:vsvers:keyval

A key value pair found in a PE vsversion info structure.

The base type for the form can be found at *file:mime:pe:vsvers:keyval*.

Properties:

name	type	doc	opts
:name	*str*	The key for the vsversion keyval pair.	Read Only: True
:value	*str*	The value for the vsversion keyval pair.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 86 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:png

The GUID of a set of mime metadata for a .png file.

The base type for the form can be found at *file:mime:png*.

Properties:

name	type	doc
:desc	*str*	MIME specific description field extracted from metadata.
:comment	*str*	MIME specific comment field extracted from metadata.
:created	*time*	MIME specific creation timestamp extracted from metadata.
:imageid	*str*	MIME specific unique identifier extracted from metadata.
:author	*ps:contact*	MIME specific contact information extracted from metadata.
:latlong	*geo:latlong*	MIME specific lat/long information extracted from metadata.
:altitude	*geo:altitude*	MIME specific altitude information extracted from metadata.
:text	*str* lower: True onespace: True	The text contained within the image.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:rtf

The GUID of a set of mime metadata for a .rtf file.

The base type for the form can be found at *file:mime:rtf* .

Properties:

name	type	doc
:guid	*guid*	The parsed GUID embedded in the .rtf file.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next pag

Table 88 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:mime:tif

The GUID of a set of mime metadata for a .tif file.

The base type for the form can be found at *file:mime:tif* .

Properties:

name	type	doc
:desc	*str*	MIME specific description field extracted from metadata.
:comment	*str*	MIME specific comment field extracted from metadata.
:created	*time*	MIME specific creation timestamp extracted from metadata.
:imageid	*str*	MIME specific unique identifier extracted from metadata.
:author	*ps:contact*	MIME specific contact information extracted from metadata.
:latlong	*geo:latlong*	MIME specific lat/long information extracted from metadata.
:altitude	*geo:altitude*	MIME specific altitude information extracted from metadata.
:text	*str* lower: True onespace: True	The text contained within the image.
:file	*file:bytes*	The file that the mime info was parsed from.
:file:offs	*int*	The optional offset where the mime info was parsed from.
:file:data	*data*	A mime specific arbitrary data structure for non-indexed data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:path

A normalized file path.

The base type for the form can be found at *file:path*.

An example of `file:path`:

- `c:/windows/system32/calc.exe`

Properties:

name	type	doc	opts
:dir	*file:path*	The parent directory.	Read Only: True
:base	*file:base*	The file base name.	Read Only: True
:base:ext	*str*	The file extension.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next

Table 90 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:string

Deprecated. Please use the edge -(refs)> it:dev:str.

The base type for the form can be found at *file:string*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file containing the string.	Read Only: True
:string	*str*	The string contained in this file:bytes node.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 91 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

file:subfile

A parent file that fully contains the specified child file.

The base type for the form can be found at *file:subfile*.

Properties:

name	type	doc	opts
:parent	*file:bytes*	The parent file containing the child file.	Read Only: True
:child	*file:bytes*	The child file contained in the parent file.	Read Only: True
:name	*file:base*	Deprecated, please use the :path property.	Deprecated: True
:path	*file:path*	The path that the parent uses to refer to the child file.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None

continues on next page

Table 92 – continued from previous page

source	verb	target	doc
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

geo:name

An unstructured place name or address.

The base type for the form can be found at *geo:name*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

geo:nloc

Records a node latitude/longitude in space-time.

The base type for the form can be found at *geo:nloc*.

Properties:

name	type	doc	opts
:ndef	*ndef*	The node with location in geospace and time.	Read Only: True
:ndef:form	*str*	The form of node referenced by the ndef.	Read Only: True
:latlong	*geo:latlong*	The latitude/longitude the node was observed.	Read Only: True
:time	*time*	The time the node was observed at location.	Read Only: True
:place	*geo:place*	The place corresponding to the latlong property.	
:loc	*loc*	The geo-political location string for the node.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 94 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

geo:place

A GUID for a geographic place.

The base type for the form can be found at *geo:place*.

Properties:

name	type	doc	opts
:name	*geo:name*	The name of the place.	
:type	*geo:place:taxonomy*	The type of place.	
:names	*array* type: *geo:name* sorted: True uniq: True	An array of alternative place names.	
:parent	*geo:place*	Deprecated. Please use a -(contains)> edge.	Deprecated: True
:desc	*str*	A long form description of the place.	
:loc	*loc*	The geo-political location string for the node.	
:address	*geo:address*	The street/mailing address for the place.	
:geojson	*geo:json*	A GeoJSON representation of the place.	
:latlong	*geo:latlong*	The lat/long position for the place.	
:bbox	*geo:bbox*	A bounding box which encompasses the place.	
:radius	*geo:dist*	An approximate radius to use for bounding box calculation.	
:photo	*file:bytes*	The image file to use as the primary image of the place.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
geo:plac	-(contains)	geo:place	The source place completely contains the target place.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
geo:place	-(contains)>	geo:place	None
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target no
it:exec:query	-(found)>	*	The target node was returned as a result of running the query
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target no
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

geo:place:taxonomy

A taxonomy of place types.

The base type for the form can be found at *geo:place:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*geo:place:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 96 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

geo:telem

A geospatial position of a node at a given time. The node should be linked via -(seenat)> edges.

The base type for the form can be found at *geo:telem*.

Properties:

name	type	doc
:time	*time*	The time that the node was at the position.
:desc	*str*	A description of the telemetry sample.
:latlong	*geo:latlong*	The latitude/longitude reading at the time.
:accuracy	*geo:dist*	The reported accuracy of the latlong telemetry reading.
:place	*geo:place*	The place which includes the latlong value.
:place:name	*geo:name*	The purported place name. Used for entity resolution.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirement	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None

continues on next

Table 97 – continued from previous page

source	verb	target	doc
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and ti
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target no
it:exec:query	-(found)>	*	The target node was returned as a result of running the query
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target no
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compr
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the targe
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

gov:cn:icp

A Chinese Internet Content Provider ID.

The base type for the form can be found at *gov:cn:icp*.

Properties:

name	type	doc
:org	*ou:org*	The org with the Internet Content Provider ID.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

gov:cn:mucd

A Chinese PLA MUCD.

The base type for the form can be found at *gov:cn:mucd*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

gov:us:cage

A Commercial and Government Entity (CAGE) code.

The base type for the form can be found at *gov:us:cage*.

Properties:

name	type	doc
:name0	*ou:name*	The name of the organization.
:name1	*str* lower: True	Name Part 1.
:street	*str* lower: True	The base string type.
:city	*str* lower: True	The base string type.
:state	*str* lower: True	The base string type.
:zip	*gov:us:zip*	A US Postal Zip Code.
:cc	*pol:iso2*	The 2 digit ISO 3166 country code.
:country	*str* lower: True	The base string type.
:phone0	*tel:phone*	A phone number.
:phone1	*tel:phone*	A phone number.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

gov:us:ssn

A US Social Security Number (SSN).

The base type for the form can be found at *gov:us:ssn*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

gov:us:zip

A US Postal Zip Code.

The base type for the form can be found at *gov:us:zip*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

graph:cluster

A generic node, used in conjunction with Edge types, to cluster arbitrary nodes to a single node in the model.

The base type for the form can be found at *graph:cluster*.

Properties:

name	type	doc
:name	*str* lower: True	A human friendly name for the cluster.
:desc	*str* lower: True	A human friendly long form description for the cluster.
:type	*str* lower: True	An optional type field used to group clusters.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 103 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

graph:edge

A generic digraph edge to show relationships outside the model.

The base type for the form can be found at *graph:edge*.

Properties:

name	type	doc	opts
:n1	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n1:form	*str*	The base string type.	Read Only: True
:n2	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n2:form	*str*	The base string type.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 104 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

graph:event

A generic event node to represent events outside the model.

The base type for the form can be found at *graph:event*.

Properties:

name	type	doc
:time	*time*	The time of the event.
:type	*str*	A arbitrary type string for the event.
:name	*str*	A name for the event.
:data	*data*	Arbitrary non-indexed msgpack data attached to the event.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

graph:node

A generic node used to represent objects outside the model.

The base type for the form can be found at *graph:node*.

Properties:

name	type	doc
:type	*str*	The type name for the non-model node.
:name	*str*	A human readable name for this record.
:data	*data*	Arbitrary non-indexed msgpack data attached to the node.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

graph:timeedge

A generic digraph time edge to show relationships outside the model.

The base type for the form can be found at *graph:timeedge*.

Properties:

name	type	doc	opts
:time	*time*	A date/time value.	Read Only: True
:n1	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n1:form	*str*	The base string type.	Read Only: True
:n2	*ndef*	The node definition type for a (form,valu) compound field.	Read Only: True
:n2:form	*str*	The base string type.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 107 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

hash:md5

A hex encoded MD5 hash.

The base type for the form can be found at *hash:md5*.

An example of `hash:md5`:

- `d41d8cd98f00b204e9800998ecf8427e`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next

Table 108 – continued from previous page

source	verb	target	doc
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compro
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

hash:sha1

A hex encoded SHA1 hash.

The base type for the form can be found at *hash:sha1*.

An example of `hash:sha1`:

- `da39a3ee5e6b4b0d3255bfef95601890afd80709`

  Source Edges:

source	verb	target	doc
*	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
*	`-(refs)>`	*	The source node contains a reference to the target node.
*	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	`-(refs)>`	*	None
`econ:purchase`	`-(acquired)>`	*	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	*	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	*	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	*	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	*	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	*	The meta:rule is designed to detect instances of the target nod
`meta:rule`	`-(matches)>`	*	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	*	The meta:source observed the target node.

continues on next

Table 109 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

hash:sha256

A hex encoded SHA256 hash.

The base type for the form can be found at *hash:sha256*.

An example of `hash:sha256`:

- `ad9f4fe922b61e674a09530831759843b1880381de686a43460a76864ca0340c`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod

continues on next

Table 110 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

hash:sha384

A hex encoded SHA384 hash.

The base type for the form can be found at *hash:sha384*.

An example of `hash:sha384`:

- `d425f1394e418ce01ed1579069a8bfaa1da8f32cf823982113ccbef531fa36bda9987f389c5af05b5e28035242efab6`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

hash:sha512

A hex encoded SHA512 hash.

The base type for the form can be found at *hash:sha512*.

An example of `hash:sha512`:

- `ca74fe2ff2d03b29339ad7d08ba21d192077fece1715291c7b43c20c9136cd132788239189f3441a87eb23ce2660aa`

  Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:asn

An Autonomous System Number (ASN).

The base type for the form can be found at *inet:asn*.

Properties:

name	type	doc
:name	*str* lower: True	The name of the organization currently responsible for the ASN.
:owner	*ou:org*	The guid of the organization currently responsible for the ASN.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:asnet4

An Autonomous System Number (ASN) and its associated IPv4 address range.

The base type for the form can be found at *inet:asnet4*.

An example of `inet:asnet4`:

- `(54959, (1.2.3.4, 1.2.3.20))`

Properties:

name	type	doc	opts
:asn	*inet:asn*	The Autonomous System Number (ASN) of the netblock.	Read Only: True
:net4	*inet:net4*	The IPv4 address range assigned to the ASN.	Read Only: True
:net4:mi	*inet:ipv4*	The first IPv4 in the range assigned to the ASN.	Read Only: True
:net4:ma:	*inet:ipv4*	The last IPv4 in the range assigned to the ASN.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.

continues on next

Table 114 – continued from previous page

source	verb	target	doc
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:asnet6

An Autonomous System Number (ASN) and its associated IPv6 address range.

The base type for the form can be found at *inet:asnet6*.

An example of `inet:asnet6`:

- `(54959, (ff::00, ff::02))`

Properties:

name	type	doc	opts
:asn	*inet:asn*	The Autonomous System Number (ASN) of the netblock.	Read Only: True
:net6	*inet:net6*	The IPv6 address range assigned to the ASN.	Read Only: True
:net6:mi	*inet:ipv6*	The first IPv6 in the range assigned to the ASN.	Read Only: True
:net6:ma:	*inet:ipv6*	The last IPv6 in the range assigned to the ASN.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:banner

A network protocol banner string presented by a server.

The base type for the form can be found at *inet:banner*.

Properties:

name	type	doc	opts
:server	*inet:server*	The server which presented the banner string.	Read Only: True
:server:ipv4	*inet:ipv4*	The IPv4 address of the server.	Read Only: True
:server:ipv6	*inet:ipv6*	The IPv6 address of the server.	Read Only: True
:server:port	*inet:port*	The network port.	Read Only: True
:text	*it:dev:str*	The banner text.	Read Only: True Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 116 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:cidr4

An IPv4 address block in Classless Inter-Domain Routing (CIDR) notation.

The base type for the form can be found at *inet:cidr4*.

An example of `inet:cidr4`:

- `1.2.3.0/24`

Properties:

name	type	doc	opts
:broadcast	*inet:ipv4*	The broadcast IP address from the CIDR notation.	Read Only: True
:mask	*int*	The mask from the CIDR notation.	Read Only: True
:network	*inet:ipv4*	The network IP address from the CIDR notation.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.

continues on next

Table 117 – continued from previous page

source	verb	target	doc
`meta:note`	`-(about)>`	*	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	*	The meta:rule is designed to detect instances of the target nod
`meta:rule`	`-(matches)>`	*	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	*	The meta:source observed the target node.
`ou:campaign`	`-(targets)>`	*	The campaign targeted the target nodes.
`ou:campaign`	`-(uses)>`	*	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	*	The contribution includes the specific node.
`ou:org`	`-(has)>`	*	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	*	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compro
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

inet:cidr6

An IPv6 address block in Classless Inter-Domain Routing (CIDR) notation.

The base type for the form can be found at *inet:cidr6*.

An example of `inet:cidr6`:

- `2001:db8::/101`

  Properties:

name	type	doc	opts
`:broadcast`	*inet:ipv6*	The broadcast IP address from the CIDR notation.	Read Only: `True`
`:mask`	*int*	The mask from the CIDR notation.	Read Only: `True`
`:network`	*inet:ipv6*	The network IP address from the CIDR notation.	Read Only: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:client

A network client address.

The base type for the form can be found at *inet:client*.

An example of `inet:client`:

- `tcp://1.2.3.4:80`

Properties:

name	type	doc	opts
`:proto`	*str* lower: `True`	The network protocol of the client.	Read Only: `True`
`:ipv4`	*inet:ipv4*	The IPv4 of the client.	Read Only: `True`
`:ipv6`	*inet:ipv6*	The IPv6 of the client.	Read Only: `True`
`:host`	*it:host*	The it:host node for the client.	Read Only: `True`
`:port`	*inet:port*	The client tcp/udp port.	

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	`*`	The meta:rule is designed to detect instances of the target nod
`meta:rule`	`-(matches)>`	`*`	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	`*`	The meta:source observed the target node.
`ou:campaign`	`-(targets)>`	`*`	The campaign targeted the target nodes.
`ou:campaign`	`-(uses)>`	`*`	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	`*`	The contribution includes the specific node.
`ou:org`	`-(has)>`	`*`	The organization is or was in possession of the target node.

continues on next

Table 119 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:a

The result of a DNS A record lookup.

The base type for the form can be found at *inet:dns:a*.

An example of `inet:dns:a`:

- `(vertex.link,1.2.3.4)`

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain queried for its DNS A record.	Read Only: `True`
:ipv4	*inet:ipv4*	The IPv4 address returned in the A record.	Read Only: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:aaaa

The result of a DNS AAAA record lookup.

The base type for the form can be found at *inet:dns:aaaa*.

An example of `inet:dns:aaaa`:

- `(vertex.link,2607:f8b0:4004:809::200e)`

  Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain queried for its DNS AAAA record.	Read Only: True
:ipv6	*inet:ipv6*	The IPv6 address returned in the AAAA record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:answer

A single answer from within a DNS reply.

The base type for the form can be found at *inet:dns:answer*.

Properties:

name	type	doc
:ttl	*int*	The base 64 bit signed integer type.
:request	*inet:dns:request*	A single instance of a DNS resolver request and optional reply info.
:a	*inet:dns:a*	The DNS A record returned by the lookup.
:ns	*inet:dns:ns*	The DNS NS record returned by the lookup.
:rev	*inet:dns:rev*	The DNS PTR record returned by the lookup.
:aaaa	*inet:dns:aaaa*	The DNS AAAA record returned by the lookup.
:rev6	*inet:dns:rev6*	The DNS PTR record returned by the lookup of an IPv6 address.
:cname	*inet:dns:cname*	The DNS CNAME record returned by the lookup.
:mx	*inet:dns:mx*	The DNS MX record returned by the lookup.
:mx:priority	*int*	The DNS MX record priority.
:soa	*inet:dns:soa*	The domain queried for its SOA record.
:txt	*inet:dns:txt*	The DNS TXT record returned by the lookup.
:time	*time*	The time that the DNS response was transmitted.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 122 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:cname

The result of a DNS CNAME record lookup.

The base type for the form can be found at *inet:dns:cname*.

An example of `inet:dns:cname`:

- `(foo.vertex.link,vertex.link)`

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain queried for its CNAME record.	Read Only: True
:cname	*inet:fqdn*	The domain returned in the CNAME record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None

continues on next

Table 123 – continued from previous page

source	verb	target	doc
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:dynreg

A dynamic DNS registration.

The base type for the form can be found at *inet:dns:dynreg*.

Properties:

name	type	doc
:fqdn	*inet:fqdn*	The FQDN registered within a dynamic DNS provider.
:provider	*ou:org*	The organization which provides the dynamic DNS FQDN.
:provider:name	*ou:name*	The name of the organization which provides the dynamic DNS FQDN.
:provider:fqdn	*inet:fqdn*	The FQDN of the organization which provides the dynamic DNS FQDN.
:contact	*ps:contact*	The contact information of the registrant.
:created	*time*	The time that the dynamic DNS registration was first created.
:client	*inet:client*	The network client address used to register the dynamic FQDN.
:client:ipv4	*inet:ipv4*	The client IPv4 address used to register the dynamic FQDN.
:client:ipv6	*inet:ipv6*	The client IPv6 address used to register the dynamic FQDN.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirement	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 124 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:mx

The result of a DNS MX record lookup.

The base type for the form can be found at *inet:dns:mx*.

An example of `inet:dns:mx`:

- `(vertex.link,mail.vertex.link)`

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain queried for its MX record.	Read Only: True
:mx	*inet:fqdn*	The domain returned in the MX record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next

Table 125 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:ns

The result of a DNS NS record lookup.

The base type for the form can be found at *inet:dns:ns*.

An example of `inet:dns:ns`:

- `(vertex.link,ns.dnshost.com)`

Properties:

name	type	doc	opts
:zone	*inet:fqdn*	The domain queried for its DNS NS record.	Read Only: True
:ns	*inet:fqdn*	The domain returned in the NS record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:query

A DNS query unique to a given client.

The base type for the form can be found at *inet:dns:query*.

An example of `inet:dns:query`:

- `(1.2.3.4, woot.com, 1)`

  Properties:

name	type	doc	opts
:client	*inet:client*	A network client address.	Read Only: True
:name	*inet:dns:nar*	A DNS query name string. Likely an FQDN but not always.	Read Only: True
:name:ipv	*inet:ipv4*	An IPv4 address.	
:name:ipv	*inet:ipv6*	An IPv6 address.	
:name:fqc	*inet:fqdn*	A Fully Qualified Domain Name (FQDN).	
:type	*int*	The base 64 bit signed integer type.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next

Table 127 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:request

A single instance of a DNS resolver request and optional reply info.

The base type for the form can be found at *inet:dns:request*.

Properties:

name	type	doc
:time	*time*	A date/time value.
:query	*inet:dns:query*	A DNS query unique to a given client.
:query:name	*inet:dns:name*	A DNS query name string. Likely an FQDN but not always.
:query:name:ipv4	*inet:ipv4*	An IPv4 address.
:query:name:ipv6	*inet:ipv6*	An IPv6 address.
:query:name:fqdn	*inet:fqdn*	A Fully Qualified Domain Name (FQDN).
:query:type	*int*	The base 64 bit signed integer type.
:server	*inet:server*	A network server address.
:reply:code	*int*	The DNS server response code.
:exe	*file:bytes*	The file containing the code that attempted the DNS lookup.
:proc	*it:exec:proc*	The process that attempted the DNS lookup.
:host	*it:host*	The host that attempted the DNS lookup.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 128 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:rev

The transformed result of a DNS PTR record lookup.

The base type for the form can be found at *inet:dns:rev*.

An example of `inet:dns:rev`:

- `(1.2.3.4,vertex.link)`

Properties:

name	type	doc	opts
:ipv4	*inet:ipv4*	The IPv4 address queried for its DNS PTR record.	Read Only: True
:fqdn	*inet:fqdn*	The domain returned in the PTR record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:rev6

The transformed result of a DNS PTR record for an IPv6 address.

The base type for the form can be found at *inet:dns:rev6*.

An example of `inet:dns:rev6`:

- `(2607:f8b0:4004:809::200e,vertex.link)`

Properties:

name	type	doc	opts
:ipv6	*inet:ipv6*	The IPv6 address queried for its DNS PTR record.	Read Only: True
:fqdn	*inet:fqdn*	The domain returned in the PTR record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next

Table 130 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:soa

The result of a DNS SOA record lookup.

The base type for the form can be found at *inet:dns:soa*.

Properties:

name	type	doc
:fqdn	*inet:fqdn*	The domain queried for its SOA record.
:ns	*inet:fqdn*	The domain (MNAME) returned in the SOA record.
:email	*inet:email*	The email address (RNAME) returned in the SOA record.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 131 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:txt

The result of a DNS MX record lookup.

The base type for the form can be found at *inet:dns:txt*.

An example of `inet:dns:txt`:

- `(hehe.vertex.link,"fancy TXT record")`

  Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain queried for its TXT record.	Read Only: True
:txt	*str*	The string returned in the TXT record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:wild:a

A DNS A wild card record and the IPv4 it resolves to.

The base type for the form can be found at *inet:dns:wild:a*.

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain containing a wild card record.	Read Only: True
:ipv4	*inet:ipv4*	The IPv4 address returned by wild card resolutions.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:dns:wild:aaaa

A DNS AAAA wild card record and the IPv6 it resolves to.

The base type for the form can be found at *inet:dns:wild:aaaa*.

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain containing a wild card record.	Read Only: True
:ipv6	*inet:ipv6*	The IPv6 address returned by wild card resolutions.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 134 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:download

An instance of a file downloaded from a server.

The base type for the form can be found at *inet:download*.

Properties:

name	type	doc
:time	*time*	The time the file was downloaded.
:fqdn	*inet:fqdn*	The FQDN used to resolve the server.
:file	*file:bytes*	The file that was downloaded.
:server	*inet:server*	The inet:addr of the server.
:server:host	*it:host*	The it:host node for the server.
:server:ipv4	*inet:ipv4*	The IPv4 of the server.
:server:ipv6	*inet:ipv6*	The IPv6 of the server.
:server:port	*inet:port*	The server tcp/udp port.
:server:proto	*str* lower: True	The server network layer protocol.
:client	*inet:client*	The inet:addr of the client.
:client:host	*it:host*	The it:host node for the client.
:client:ipv4	*inet:ipv4*	The IPv4 of the client.
:client:ipv6	*inet:ipv6*	The IPv6 of the client.
:client:port	*inet:port*	The client tcp/udp port.
:client:proto	*str* lower: True	The client network layer protocol.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:egress

A host using a specific network egress client address.

The base type for the form can be found at *inet:egress*.

Properties:

name	type	doc
:host	*it:host*	The host that used the network egress.
:client	*inet:client*	The client address the host used as a network egress.
:client:ipv4	*inet:ipv4*	The client IPv4 address the host used as a network egress.
:client:ipv6	*inet:ipv6*	The client IPv6 address the host used as a network egress.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:email

An e-mail address.

The base type for the form can be found at *inet:email*.

Properties:

name	type	doc	opts
:user	*inet:user*	The username of the email address.	Read Only: True
:fqdn	*inet:fqdn*	The domain of the email address.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 137 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:email:header

A unique email message header.

The base type for the form can be found at *inet:email:header*.

Properties:

name	type	doc	opts
:name	*inet:email:header:name*	The name of the email header.	Read Only: True
:value	*str*	The value of the email header.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 138 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:email:message

An individual email message delivered to an inbox.

The base type for the form can be found at *inet:email:message*.

Properties:

name	type	doc	opts
:to	*inet:email*	The email address of the recipient.	
:from	*inet:email*	The email address of the sender.	
:replyto	*inet:email*	The email address parsed from the "reply-to" header.	
:cc	*array* type: *inet:email* uniq: True sorted: True	Email addresses parsed from the "cc" header.	
:subject	*str*	The email message subject parsed from the "subject" header.	
:body	*str*	The body of the email message.	Display: {'hint': 'text'}
:date	*time*	The time the email message was delivered.	
:bytes	*file:bytes*	The file bytes which contain the email message.	
:headers	*array* type: *inet:email:header*	An array of email headers from the message.	
:received:from:ipv	*inet:ipv4*	The sending SMTP server IPv4, potentially from the Received: header.	
:received:from:ipv	*inet:ipv6*	The sending SMTP server IPv6, potentially from the Received: header.	
:received:from:fqd	*inet:fqdn*	The sending server FQDN, potentially from the Received: header.	
:flow	*inet:flow*	The inet:flow which delivered the message.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:email:message:attachment

A file which was attached to an email message.

The base type for the form can be found at *inet:email:message:attachment*.

Properties:

name	type	doc	opts
:message	*inet:email:message*	The message containing the attached file.	Read Only: True
:file	*file:bytes*	The attached file.	Read Only: True
:name	*file:base*	The name of the attached file.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 140 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:email:message:link

A url/link embedded in an email message.

The base type for the form can be found at *inet:email:message:link*.

Properties:

name	type	doc	opts
:message	*inet:email:messag*	The message containing the embedded link.	Read Only: True
:url	*inet:url*	The url contained within the email message.	Read Only: True
:text	*str*	The displayed hyperlink text if it was not the raw URL.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 141 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:flow

An individual network connection between a given source and destination.

The base type for the form can be found at *inet:flow*.

Properties:

name	type	doc	opts
:time	*time*	The time the network connection was initiated.	
:duration	*int*	The duration of the flow in seconds.	
:from	*guid*	The ingest source file/iden. Used for reparsing.	
:dst	*inet:server*	The destination address / port for a connection.	
:dst:ipv4	*inet:ipv4*	The destination IPv4 address.	
:dst:ipv6	*inet:ipv6*	The destination IPv6 address.	
:dst:port	*inet:port*	The destination port.	
:dst:proto	*str* lower: True	The destination protocol.	

continues on next page

Table 142 – continued from previous page

name	type	doc	opts
`:dst:host`	*it:host*	The guid of the destination host.	
`:dst:proc`	*it:exec:proc*	The guid of the destination process.	
`:dst:exe`	*file:bytes*	The file (executable) that received the connection.	
`:dst:txcount`	*int*	The number of packets sent by the destination host.	
`:dst:txbytes`	*int*	The number of bytes sent by the destination host.	
`:dst:handshake`	*str*	A text representation of the initial handshake sent by the server.	Display: `{'hint': 'text'}`
`:src`	*inet:client*	The source address / port for a connection.	
`:src:ipv4`	*inet:ipv4*	The source IPv4 address.	
`:src:ipv6`	*inet:ipv6*	The source IPv6 address.	
`:src:port`	*inet:port*	The source port.	
`:src:proto`	*str* lower: `True`	The source protocol.	
`:src:host`	*it:host*	The guid of the source host.	
`:src:proc`	*it:exec:proc*	The guid of the source process.	
`:src:exe`	*file:bytes*	The file (executable) that created the connection.	
`:src:txcount`	*int*	The number of packets sent by the source host.	
`:src:txbytes`	*int*	The number of bytes sent by the source host.	
`:tot:txcount`	*int*	The number of packets sent in both directions.	
`:tot:txbytes`	*int*	The number of bytes sent in both directions.	

continues on next page

Table 142 – continued from previous page

name	type	doc	opts
:src:handshake	*str*	A text representation of the initial handshake sent by the client.	Display: {'hint': 'text'}
:dst:cpes	*array* type: *it:sec:cpe* uniq: True sorted: True	An array of NIST CPEs identified on the destination host.	
:dst:softnames	*array* type: *it:prod:softname* uniq: True sorted: True	An array of software names identified on the destination host.	
:src:cpes	*array* type: *it:sec:cpe* uniq: True sorted: True	An array of NIST CPEs identified on the source host.	
:src:softnames	*array* type: *it:prod:softname* uniq: True sorted: True	An array of software names identified on the source host.	
:ip:proto	*int* min: 0 max: 255	The IP protocol number of the flow.	
:ip:tcp:flags	*int* min: 0 max: 255	An aggregation of observed TCP flags commonly provided by flow APIs.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:src:ssl:cert	*crypto:x509:cert*	The x509 certificate sent by the client as part of an SSL/TLS negotiation.	

continues on next page

Table 142 – continued from previous page

name	type	doc	opts
:dst:ssl:cert	*crypto:x509:cert*	The x509 certificate sent by the server as part of an SSL/TLS negotiation.	
:src:rdp:hostname	*it:hostname*	The hostname sent by the client as part of an RDP session setup.	
:src:rdp:keyboard:	*str* lower: True onespace: True	The keyboard layout sent by the client as part of an RDP session setup.	
:src:ssh:key	*crypto:key*	The key sent by the client as part of an SSH session setup.	
:dst:ssh:key	*crypto:key*	The key sent by the server as part of an SSH session setup.	
:raw	*data*	A raw record used to create the flow which may contain additional protocol details.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 143 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:fqdn

A Fully Qualified Domain Name (FQDN).

The base type for the form can be found at *inet:fqdn*.

An example of `inet:fqdn`:

- `vertex.link`

Properties:

name	type	doc	opts
:domain	*inet:fqdn*	The parent domain for the FQDN.	Read Only: `True`
:host	*str* lower: `True`	The host part of the FQDN.	Read Only: `True`
:issuffix	*bool*	True if the FQDN is considered a suffix.	
:iszone	*bool*	True if the FQDN is considered a zone.	
:zone	*inet:fqdn*	The zone level parent for this FQDN.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:group

A group name string.

The base type for the form can be found at *inet:group*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:cookie

An individual HTTP cookie string.

The base type for the form can be found at *inet:http:cookie*.

An example of `inet:http:cookie`:

- `PHPSESSID=el4ukv0kqbvoirg7nkp4dncpk3`

Properties:

name	type	doc
:name	*str*	The name of the cookie preceding the equal sign.
:value	*str*	The value of the cookie after the equal sign if present.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next

Table 146 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:param

An HTTP request path query parameter.

The base type for the form can be found at *inet:http:param*.

Properties:

name	type	doc	opts
:name	*str* lower: True	The name of the HTTP query parameter.	Read Only: True
:value	*str*	The value of the HTTP query parameter.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 147 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:request

A single HTTP request.

The base type for the form can be found at *inet:http:request*.

Properties:

name	type	doc
:method	*str*	The HTTP request method string.
:path	*str*	The requested HTTP path (without query parameters).
:url	*inet:url*	The reconstructed URL for the request if known.
:query	*str*	The HTTP query string which optionally follows the path.
:headers	*array* type: *inet:http:request:header*	An array of HTTP headers from the request.
:body	*file:bytes*	The body of the HTTP request.

continues on next page

Table 148 – continued from previous page

name	type	doc
:referer	*inet:url*	The referer URL parsed from the "Referer:" header in the request.
:cookies	*array* type: *inet:http:cookie* sorted: True uniq: True	An array of HTTP cookie values parsed from the "Cookies:" header in the request.
:response:time	*time*	A date/time value.
:response:code	*int*	The base 64 bit signed integer type.
:response:reason	*str*	The base string type.
:response:headers	*array* type: *inet:http:response:header*	An array of HTTP headers from the response.
:response:body	*file:bytes*	The file bytes type with SHA256 based primary property.
:session	*inet:http:session*	The HTTP session this request was part of.
:flow	*inet:flow*	The raw inet:flow containing the request.
:client	*inet:client*	The inet:addr of the client.
:client:ipv4	*inet:ipv4*	The server IPv4 address that the request was sent from.
:client:ipv6	*inet:ipv6*	The server IPv6 address that the request was sent from.
:client:host	*it:host*	The host that the request was sent from.
:server	*inet:server*	The inet:addr of the server.
:server:ipv4	*inet:ipv4*	The server IPv4 address that the request was sent to.
:server:ipv6	*inet:ipv6*	The server IPv6 address that the request was sent to.
:server:port	*inet:port*	The server port that the request was sent to.
:server:host	*it:host*	The host that the request was sent to.
:exe	*file:bytes*	The executable file which caused the activity.
:proc	*it:exec:proc*	The host process which caused the activity.
:thread	*it:exec:thread*	The host thread which caused the activity.
:host	*it:host*	The host on which the activity occurred.

continues on next page

Table 148 – continued from previous page

name	type	doc
:time	*time*	The time that the activity started.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

continues on next page

Table 149 – continued from previous page

source	verb	target	doc
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:request:header

An HTTP request header.

The base type for the form can be found at *inet:http:request:header*.

Properties:

name	type	doc	opts
:name	*inet:http:header:name*	The name of the HTTP request header.	Read Only: True
:value	*str*	The value of the HTTP request header.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 150 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:response:header

An HTTP response header.

The base type for the form can be found at *inet:http:response:header*.

Properties:

name	type	doc	opts
:name	*inet:http:header:name*	The name of the HTTP response header.	Read Only: True
:value	*str*	The value of the HTTP response header.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 151 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:http:session

An HTTP session.

The base type for the form can be found at *inet:http:session*.

Properties:

name	type	doc
:contact	*ps:contact*	The ps:contact which owns the session.
:cookies	*array* type: *inet:http:cookie* sorted: True uniq: True	An array of cookies used to identify this specific session.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:iface

A network interface with a set of associated protocol addresses.

The base type for the form can be found at *inet:iface*.

Properties:

name	type	doc
:host	*it:host*	The guid of the host the interface is associated with.
:network	*it:network*	The guid of the it:network the interface connected to.
:type	*str* lower: True	The free-form interface type.
:mac	*inet:mac*	The ethernet (MAC) address of the interface.
:ipv4	*inet:ipv4*	The IPv4 address of the interface.
:ipv6	*inet:ipv6*	The IPv6 address of the interface.
:phone	*tel:phone*	The telephone number of the interface.
:wifi:ssid	*inet:wifi:ssid*	The wifi SSID of the interface.
:wifi:bssid	*inet:mac*	The wifi BSSID of the interface.
:adid	*it:adid*	An advertising ID associated with the interface.
:mob:imei	*tel:mob:imei*	The IMEI of the interface.
:mob:imsi	*tel:mob:imsi*	The IMSI of the interface.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 153 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:ipv4

An IPv4 address.

The base type for the form can be found at *inet:ipv4*.

An example of `inet:ipv4`:

- `1.2.3.4`

Properties:

name	type	doc
:asn	*inet:asn*	The ASN to which the IPv4 address is currently assigned.
:latlong	*geo:latlong*	The best known latitude/longitude for the node.
:loc	*loc*	The geo-political location string for the IPv4.
:place	*geo:place*	The geo:place associated with the latlong property.
:type	*str*	The type of IP address (e.g., private, multicast, etc.).
:dns:rev	*inet:fqdn*	The most current DNS reverse lookup for the IPv4.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
inet:whois:iprec	-(ipwhois)>	inet:ipv4	The source IP whois record describes the target IPv4 add
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the targe
it:exec:query	-(found)>	*	The target node was returned as a result of running the q
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the targe
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target no
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the co
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the t
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target node
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:ipv6

An IPv6 address.

The base type for the form can be found at *inet:ipv6*.

An example of `inet:ipv6`:

- `2607:f8b0:4004:809::200e`

Properties:

name	type	doc
:asn	*inet:asn*	The ASN to which the IPv6 address is currently assigned.
:ipv4	*inet:ipv4*	The mapped ipv4.
:latlong	*geo:latlong*	The last known latitude/longitude for the node.
:place	*geo:place*	The geo:place associated with the latlong property.
:dns:rev	*inet:fqdn*	The most current DNS reverse lookup for the IPv6.
:loc	*loc*	The geo-political location string for the IPv6.
:type	*str*	The type of IP address (e.g., private, multicast, etc.).
:scope	*str* enums: reserved, interface-local, link-local, realm-local, admin-local, site-local, organization-local, global,unassigned	The IPv6 scope of the address (e.g., global, link-local, etc.).

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
inet:whois:iprec	-(ipwhois)>	inet:ipv6	The source IP whois record describes the target IPv6 add
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the targe
it:exec:query	-(found)>	*	The target node was returned as a result of running the q
meta:note	-(about)>	*	The meta:note is about the target node.

continues on

Table 155 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the targe
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target no
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the co
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the t
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target node
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:mac

A 48-bit Media Access Control (MAC) address.

The base type for the form can be found at *inet:mac*.

An example of `inet:mac`:

- `aa:bb:cc:dd:ee:ff`

Properties:

name	type	doc
:vendor	*str*	The vendor associated with the 24-bit prefix of a MAC address.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:passwd

A password string.

The base type for the form can be found at *inet:passwd*.

Properties:

name	type	doc	opts
:md5	*hash:md5*	The MD5 hash of the password.	Read Only: True
:sha1	*hash:sha1*	The SHA1 hash of the password.	Read Only: True
:sha256	*hash:sha256*	The SHA256 hash of the password.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:proto

A network protocol name.

The base type for the form can be found at *inet:proto*.

Properties:

name	type	doc
:port	*inet:port*	The default port this protocol typically uses if applicable.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.

continues on next page

Table 158 – continued from previous page

source	verb	target	doc
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:rfc2822:addr

An RFC 2822 Address field.

The base type for the form can be found at *inet:rfc2822:addr*.

An example of `inet:rfc2822:addr`:

- `"Visi Kenshoto" <visi@vertex.link>`

Properties:

name	type	doc	opts
:name	*ps:name*	The name field parsed from an RFC 2822 address string.	Read Only: True
:email	*inet:email*	The email field parsed from an RFC 2822 address string.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next

Table 159 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:search:query

An instance of a search query issued to a search engine.

The base type for the form can be found at *inet:search:query*.

Properties:

name	type	doc	opts
:text	*str*	The search query text.	Display: {'hint': 'text'}
:time	*time*	The time the web search was issued.	
:acct	*inet:web:acct*	The account that the query was issued as.	
:host	*it:host*	The host that issued the query.	
:engine	*str* lower: True	A simple name for the search engine used.	Example: google
:request	*inet:http:request*	The HTTP request used to issue the query.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:search:result

A single result from a web search.

The base type for the form can be found at *inet:search:result*.

Properties:

name	type	doc
:query	*inet:search:query*	The search query that produced the result.
:title	*str* lower: True	The title of the matching web page.
:rank	*int*	The rank/order of the query result.
:url	*inet:url*	The URL hosting the matching content.
:text	*str* lower: True	Extracted/matched text from the matched content.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 161 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:server

A network server address.

The base type for the form can be found at *inet:server*.

An example of `inet:server`:

- `tcp://1.2.3.4:80`

Properties:

name	type	doc	opts
:proto	*str* lower: True	The network protocol of the server.	Read Only: True
:ipv4	*inet:ipv4*	The IPv4 of the server.	Read Only: True
:ipv6	*inet:ipv6*	The IPv6 of the server.	Read Only: True
:host	*it:host*	The it:host node for the server.	Read Only: True
:port	*inet:port*	The server tcp/udp port.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:servfile

A file hosted on a server for access over a network protocol.

The base type for the form can be found at *inet:servfile*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file hosted by the server.	Read Only: True
:server	*inet:server*	The inet:addr of the server.	Read Only: True
:server:proto	*str* lower: True	The network protocol of the server.	Read Only: True
:server:ipv4	*inet:ipv4*	The IPv4 of the server.	Read Only: True
:server:ipv6	*inet:ipv6*	The IPv6 of the server.	Read Only: True
:server:host	*it:host*	The it:host node for the server.	Read Only: True
:server:port	*inet:port*	The server tcp/udp port.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 163 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:ssl:cert

Deprecated. Please use inet:tls:servercert or inet:tls:clientcert.

The base type for the form can be found at *inet:ssl:cert*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file bytes for the SSL certificate.	Read Only: True
:server	*inet:server*	The server that presented the SSL certificate.	Read Only: True
:server:ipv4	*inet:ipv4*	The SSL server IPv4 address.	Read Only: True
:server:ipv6	*inet:ipv6*	The SSL server IPv6 address.	Read Only: True
:server:port	*inet:port*	The SSL server listening port.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None

continues on next page

Table 164 – continued from previous page

source	verb	target	doc
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:ssl:jarmhash

A TLS JARM fingerprint hash.

The base type for the form can be found at *inet:ssl:jarmhash*.

Properties:

name	type	doc	opts
:ciphers	*str* lower: True strip: True regex: `^[0-9a-f]{30}$`	The encoded cipher and TLS version of the server.	Read Only: True
:extensions	*str* lower: True strip: True regex: `^[0-9a-f]{32}$`	The truncated SHA256 of the TLS server extensions.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 165 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:ssl:jarmsample

A JARM hash sample taken from a server.

The base type for the form can be found at *inet:ssl:jarmsample*.

Properties:

name	type	doc	opts
:jarmhash	*inet:ssl:jarmhas*	The JARM hash computed from the server responses.	Read Only: True
:server	*inet:server*	The server that was sampled to compute the JARM hash.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 166 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tls:clientcert

An x509 certificate sent by a client for TLS.

The base type for the form can be found at *inet:tls:clientcert*.

An example of `inet:tls:clientcert`:

- `(1.2.3.4:443, 3fdf364e081c14997b291852d1f23868)`

Properties:

name	type	doc	opts
:client	*inet:client*	The client associated with the x509 certificate.	Read Only: True
:cert	*crypto:x509:cer*	The x509 certificate sent by the client.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tls:handshake

An instance of a TLS handshake between a server and client.

The base type for the form can be found at *inet:tls:handshake*.

Properties:

name	type	doc
:time	*time*	The time the handshake was initiated.
:flow	*inet:flow*	The raw inet:flow associated with the handshake.
:server	*inet:server*	The TLS server during the handshake.
:server:cert	*crypto:x509:cei*	The x509 certificate sent by the server during the handshake.
:server:fingerprint:]	*hash:md5*	The JA3S finger of the server.
:client	*inet:client*	The TLS client during the handshake.
:client:cert	*crypto:x509:cei*	The x509 certificate sent by the client during the handshake.
:client:fingerprint:]	*hash:md5*	The JA3 fingerprint of the client.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 168 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tls:ja3:sample

A JA3 sample taken from a client.

The base type for the form can be found at *inet:tls:ja3:sample*.

Properties:

name	type	doc	opts
:client	*inet:client*	The client that was sampled to produce the JA3 hash.	Read Only: True
:ja3	*hash:md5*	The JA3 hash computed from the client's TLS hello packet.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 169 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tls:ja3s:sample

A JA3 sample taken from a server.

The base type for the form can be found at *inet:tls:ja3s:sample*.

Properties:

name	type	doc	opts
:server	*inet:server*	The server that was sampled to produce the JA3S hash.	Read Only: True
:ja3s	*hash:md5*	The JA3S hash computed from the server's TLS hello packet.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 170 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tls:servercert

An x509 certificate sent by a server for TLS.

The base type for the form can be found at *inet:tls:servercert*.

An example of `inet:tls:servercert`:

- `(1.2.3.4:443, c7437790af01ae1bb2f8f3b684c70bf8)`

Properties:

name	type	doc	opts
:server	*inet:server*	The server associated with the x509 certificate.	Read Only: True
:cert	*crypto:x509:cer*	The x509 certificate sent by the server.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tunnel

A specific sequence of hosts forwarding connections such as a VPN or proxy.

The base type for the form can be found at *inet:tunnel*.

Properties:

name	type	doc
:anon	*bool*	Indicates that this tunnel provides anonymization.
:type	*inet:tunnel:type:taxonomy*	The type of tunnel such as vpn or proxy.
:ingress	*inet:server*	The server where client traffic enters the tunnel.
:egress	*inet:server*	The server where client traffic leaves the tunnel.
:operator	*ps:contact*	The contact information for the tunnel operator.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 172 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:tunnel:type:taxonomy

A taxonomy of network tunnel types.

The base type for the form can be found at *inet:tunnel:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*inet:tunnel:type:taxono*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 173 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:url

A Universal Resource Locator (URL).

The base type for the form can be found at *inet:url*.

An example of `inet:url`:

- `http://www.woot.com/files/index.html`

  Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The fqdn used in the URL (e.g., http://www.woot.com/page.html).	Read Only: True
:ipv4	*inet:ipv4*	The IPv4 address used in the URL (e.g., http://1.2.3.4/page.html).	Read Only: True
:ipv6	*inet:ipv6*	The IPv6 address used in the URL.	Read Only: True
:passwd	*inet:passwd*	The optional password used to access the URL.	Read Only: True
:base	*str*	The base scheme, user/pass, fqdn, port and path w/o parameters.	Read Only: True
:path	*str*	The path in the URL w/o parameters.	Read Only: True
:params	*str*	The URL parameter string.	Read Only: True
:port	*inet:port*	The port of the URL. URLs prefixed with http will be set to port 80 and URLs prefixed with https will be set to port 443 unless otherwise specified.	Read Only: True
:proto	*str* lower: True	The protocol in the URL.	Read Only: True
:user	*inet:user*	The optional username used to access the URL.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:url:mirror

A URL mirror site.

The base type for the form can be found at *inet:url:mirror*.

Properties:

name	type	doc	opts
:of	*inet:url*	The URL being mirrored.	Read Only: True
:at	*inet:url*	The URL of the mirror.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:urlfile

A file hosted at a specific Universal Resource Locator (URL).

The base type for the form can be found at *inet:urlfile*.

Properties:

name	type	doc	opts
:url	*inet:url*	The URL where the file was hosted.	Read Only: True
:file	*file:bytes*	The file that was hosted at the URL.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 176 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:urlredir

A URL that redirects to another URL, such as via a URL shortening service or an HTTP 302 response.

The base type for the form can be found at *inet:urlredir*.

An example of `inet:urlredir`:

- `(http://foo.com/,http://bar.com/)`

Properties:

name	type	doc	opts
:src	*inet:url*	The original/source URL before redirect.	Read Only: True
:src:fqdn	*inet:fqdn*	The FQDN within the src URL (if present).	Read Only: True
:dst	*inet:url*	The redirected/destination URL.	Read Only: True
:dst:fqdn	*inet:fqdn*	The FQDN within the dst URL (if present).	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next

Table 177 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:user

A username string.

The base type for the form can be found at *inet:user*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 178 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:acct

An account with a given Internet-based site or service.

The base type for the form can be found at *inet:web:acct*.

An example of `inet:web:acct`:

- `twitter.com/invisig0th`

  Properties:

name	type	doc	opts
:avatar	*file:bytes*	The file representing the avatar (e.g., profile picture) for the account.	
:banner	*file:bytes*	The file representing the banner for the account.	
:dob	*time*	A self-declared date of birth for the account (if the account belongs to a person).	
:email	*inet:email*	The email address associated with the account.	
:linked:accts	*array* type: *inet:web:acct* uniq: `True` sorted: `True`	Linked accounts specified in the account profile.	
:latlong	*geo:latlong*	The last known latitude/longitude for the node.	
:place	*geo:place*	The geo:place associated with the latlong property.	
:loc	*loc*	A self-declared location for the account.	
:name	*inet:user*	The localized name associated with the account (may be different from the account identifier, e.g., a display name).	
:name:en	*inet:user*	The English version of the name associated with the (may be different from the account identifier, e.g., a display name).	Deprecated: `True`
:aliases	*array* type: *inet:user* uniq: `True` sorted: `True`	An array of alternate names for the user.	
:occupation	*str* lower: `True`	A self-declared occupation for the account.	
:passwd	*inet:passwd*	The current pass-	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:action

An instance of an account performing an action at an Internet-based site or service.

The base type for the form can be found at *inet:web:action*.

Properties:

name	type	doc
:act	*str* lower: True strip: True	The action performed by the account.
:acct	*inet:web:acct*	The web account associated with the action.
:acct:site	*inet:fqdn*	The site or service associated with the account.
:acct:user	*inet:user*	The unique identifier for the account.
:time	*time*	The date and time the account performed the action.
:client	*inet:client*	The source client address of the action.
:client:ipv4	*inet:ipv4*	The source IPv4 address of the action.
:client:ipv6	*inet:ipv6*	The source IPv6 address of the action.
:loc	*loc*	The location of the user executing the web action.
:latlong	*geo:latlong*	The latlong of the user when executing the web action.
:place	*geo:place*	The geo:place of the user when executing the web action.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 180 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:attachment

An instance of a file being sent to a web service by an account.

The base type for the form can be found at *inet:web:attachment*.

Properties:

name	type	doc	opts
:acct	*inet:web:acct*	The account that uploaded the file.	
:post	*inet:web:post*	The optional web post that the file was attached to.	
:mesg	*inet:web:mes*	The optional web message that the file was attached to.	
:proto	*inet:proto*	The protocol used to transmit the file to the web service.	Example: `https`
:interactiv	*bool*	Set to true if the upload was interactive. False if automated.	
:file	*file:bytes*	The file that was sent.	
:name	*file:path*	The name of the file at the time it was sent.	
:time	*time*	The time the file was sent.	
:client	*inet:client*	The client address which initiated the upload.	
:client:ipv	*inet:ipv4*	The IPv4 address of the client that initiated the upload.	
:client:ipv	*inet:ipv6*	The IPv6 address of the client that initiated the upload.	
:place	*geo:place*	The place the file was sent from.	
:place:loc	*loc*	The geopolitical location that the file was sent from.	
:place:name	*geo:name*	The reported name of the place that the file was sent from.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 181 – continued from previous page

source	verb	target	doc
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

inet:web:channel

A channel within a web service or instance such as slack or discord.

The base type for the form can be found at *inet:web:channel*.

Properties:

name	type	doc	opts
:url	*inet:url*	The primary URL used to identify the channel.	Example: `https://app.slack.com/client/T2XK1223Y/C2XHHNDS7`
:id	*str* strip: `True`	The operator specified ID of this channel.	Example: `C2XHHNDS7`
:instance	*inet:web:instance*	The instance which contains the channel.	
:name	*str* strip: `True`	The visible name of the channel.	Example: `general`
:topic	*str* strip: `True`	The visible topic of the channel.	Example: `Synapse Discussion - Feel free to invite others!`
:created	*time*	The time the channel was created.	
:creator	*inet:web:acct*	The account which created the channel.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 182 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:chprofile

A change to a web account. Used to capture historical properties associated with an account, as opposed to current data in the inet:web:acct node.

The base type for the form can be found at *inet:web:chprofile*.

Properties:

name	type	doc
:acct	*inet:wel*	The web account associated with the change.
:acct:s	*inet:fqd*	The site or service associated with the account.
:acct:u	*inet:use*	The unique identifier for the account.
:client	*inet:clie*	The source address used to make the account change.
:client	*inet:ipv*	The source IPv4 address used to make the account change.
:client	*inet:ipv*	The source IPv6 address used to make the account change.
:time	*time*	The date and time when the account change occurred.
:pv	*node-prop*	The prop=valu of the account property that was changed. Valu should be the old / original value, while the new value should be updated on the inet:web:acct form.
:pv:prc	*str*	The property that was changed.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:file

A file posted by a web account.

The base type for the form can be found at *inet:web:file*.

Properties:

name	type	doc	opts
:acct	*inet:web:acc*	The account that owns or is associated with the file.	Read Only: True
:acct:site	*inet:fqdn*	The site or service associated with the account.	Read Only: True
:acct:user	*inet:user*	The unique identifier for the account.	Read Only: True
:file	*file:bytes*	The file owned by or associated with the account.	Read Only: True
:name	*file:base*	The name of the file owned by or associated with the account.	
:posted	*time*	Deprecated. Instance data belongs on inet:web:attachment.	Deprecated: True
:client	*inet:client*	Deprecated. Instance data belongs on inet:web:attachment.	Deprecated: True
:client:ipv	*inet:ipv4*	Deprecated. Instance data belongs on inet:web:attachment.	Deprecated: True
:client:ipv	*inet:ipv6*	Deprecated. Instance data belongs on inet:web:attachment.	Deprecated: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 184 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:follows

A web account follows or is connected to another web account.

The base type for the form can be found at *inet:web:follows*.

Properties:

name	type	doc	opts
:follower	*inet:web:acct*	The account following an account.	Read Only: True
:followee	*inet:web:acct*	The account followed by an account.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 185 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:group

A group hosted within or registered with a given Internet-based site or service.

The base type for the form can be found at *inet:web:group*.

An example of `inet:web:group`:

- `somesite.com/mycoolgroup`

  Properties:

name	type	doc	opts
:site	*inet:fqdn*	The site or service associated with the group.	Read Only: True
:id	*inet:group*	The site-specific unique identifier for the group (may be different from the common name or display name).	Read Only: True
:name	*inet:group*	The localized name associated with the group (may be different from the account identifier, e.g., a display name).	
:aliases	*array* type: *inet:group* uniq: True sorted: True	An array of alternate names for the group.	
:name:en	*inet:group*	The English version of the name associated with the group (may be different from the localized name).	Deprecated: True
:url	*inet:url*	The service provider URL where the group is hosted.	
:avatar	*file:bytes*	The file representing the avatar (e.g., profile picture) for the group.	
:desc	*str*	The text of the description of the group.	
:webpage	*inet:url*	A related URL specified by the group (e.g., primary web site, etc.).	
:loc	*str* lower: True	A self-declared location for the group.	
:latlong	*geo:latlong*	The last known latitude/longitude for the node.	
:place	*geo:place*	The geo:place associated with the latlong property.	
:signup	*time*	The date and time the group was cre-	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:hashtag

A hashtag used in a web post.

The base type for the form can be found at *inet:web:hashtag*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:instance

An instance of a web service such as slack or discord.

The base type for the form can be found at *inet:web:instance*.

Properties:

name	type	doc	opts
:url	*inet:url*	The primary URL used to identify the instance.	Example: `https://app.slack.com/client/T2XK1223Y`
:id	*str* strip: `True`	The operator specified ID of this instance.	Example: `T2XK1223Y`
:name	*str* strip: `True`	The visible name of the instance.	Example: `vertex synapse`
:created	*time*	The time the instance was created.	
:creator	*inet:web:acct*	The account which created the instance.	
:owner	*ou:org*	The organization which created the instance.	
:owner:fqdn	*inet:fqdn*	The FQDN of the organization which created the instance. Used for entity resolution.	Example: `vertex.link`
:owner:name	*ou:name*	The name of the organization which created the instance. Used for entity resolution.	Example: `the vertex project, llc.`
:operator	*ou:org*	The organization which operates the instance.	
:operator:name	*ou:name*	The name of the organization which operates the instance. Used for entity resolution.	Example: `slack`
:operator:fqdn	*inet:fqdn*	The FQDN of the organization which operates the instance. Used for entity resolution.	Example: `slack.com`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:logon

An instance of an account authenticating to an Internet-based site or service.

The base type for the form can be found at *inet:web:logon*.

Properties:

name	type	doc
:acct	*inet:web:acct*	The web account associated with the logon event.
:acct:site	*inet:fqdn*	The site or service associated with the account.
:acct:user	*inet:user*	The unique identifier for the account.
:time	*time*	The date and time the account logged into the service.
:client	*inet:client*	The source address of the logon.
:client:ipv4	*inet:ipv4*	The source IPv4 address of the logon.
:client:ipv6	*inet:ipv6*	The source IPv6 address of the logon.
:logout	*time*	The date and time the account logged out of the service.
:loc	*loc*	The location of the user executing the logon.
:latlong	*geo:latlong*	The latlong of the user executing the logon.
:place	*geo:place*	The geo:place of the user executing the logon.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 189 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:memb

Deprecated. Please use inet:web:member.

The base type for the form can be found at *inet:web:memb*.

Properties:

name	type	doc	opts
:acct	*inet:web:acct*	The account that is a member of the group.	Read Only: True
:group	*inet:web:group*	The group that the account is a member of.	Read Only: True
:title	*str* lower: True	The title or status of the member (e.g., admin, new member, etc.).	
:joined	*time*	The date / time the account joined the group.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:member

Represents a web account membership in a channel or group.

The base type for the form can be found at *inet:web:member*.

Properties:

name	type	doc
:acct	*inet:web:acct*	The account that is a member of the group or channel.
:group	*inet:web:group*	The group that the account is a member of.
:channel	*inet:web:channel*	The channel that the account is a member of.
:added	*time*	The date / time the account was added to the group or channel.
:removed	*time*	The date / time the account was removed from the group or channel.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:mesg

A message sent from one web account to another web account or channel.

The base type for the form can be found at *inet:web:mesg*.

An example of `inet:web:mesg`:

- `((twitter.com, invisig0th), (twitter.com, gobbles), 20041012130220)`

Properties:

name	type	doc	opts
`:from`	*inet:web:a*	The web account that sent the message.	Read Only: `True`
`:to`	*inet:web:a*	The web account that received the message.	Read Only: `True`
`:client`	*inet:client*	The source address of the message.	
`:client`	*inet:ipv4*	The source IPv4 address of the message.	
`:client`	*inet:ipv6*	The source IPv6 address of the message.	
`:time`	*time*	The date and time at which the message was sent.	Read Only: `True`
`:url`	*inet:url*	The URL where the message is posted / visible.	
`:text`	*str*	The text of the message.	Display: `{'hint': 'text'}`
`:delete`	*bool*	The message was deleted.	
`:file`	*file:bytes*	The file attached to or sent with the message.	
`:place`	*geo:place*	The place that the message was reportedly sent from.	
`:place:`	*geo:name*	The name of the place that the message was reportedly sent from. Used for entity resolution.	
`:instan`	*inet:web:i*	The instance where the message was sent.	

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.

continues on next

Table 192 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:post

A post made by a web account.

The base type for the form can be found at *inet:web:post*.

Properties:

name	type	doc	opts
`:acct`	*inet:web:acct*	The web account that made the post.	
`:acct:site`	*inet:fqdn*	The site or service associated with the account.	
`:client`	*inet:client*	The source address of the post.	
`:client:ipv4`	*inet:ipv4*	The source IPv4 address of the post.	
`:client:ipv6`	*inet:ipv6*	The source IPv6 address of the post.	
`:acct:user`	*inet:user*	The unique identifier for the account.	
`:text`	*str*	The text of the post.	Display: `{'hint': 'text'}`
`:time`	*time*	The date and time that the post was made.	
`:deleted`	*bool*	The message was deleted by the poster.	
`:url`	*inet:url*	The URL where the post is published / visible.	
`:file`	*file:bytes*	The file that was attached to the post.	
`:replyto`	*inet:web:post*	The post that this post is in reply to.	
`:repost`	*inet:web:post*	The original post that this is a repost of.	
`:hashtags`	*array* type: *inet:web:hashtag* uniq: `True` sorted: `True` split: `,`	Hashtags mentioned within the post.	
`:mentions:users`	*array* type: *inet:web:acct* uniq: `True` sorted: `True` split: `,`	Accounts mentioned within the post.	
`:mentions:groups`	*array* type: *inet:web:group* uniq: `True` sorted: `True` split: `,`	Groups mentioned within the post.	
`:loc`	*loc*	The location that the post was reportedly sent from.	
`:place`	*geo:place*	The place that the post was reportedly sent	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:web:post:link

A link contained within post text.

The base type for the form can be found at *inet:web:post:link*.

Properties:

name	type	doc
:post	*inet:web:post*	The post containing the embedded link.
:url	*inet:url*	The url that the link forwards to.
:text	*str*	The displayed hyperlink text if it was not the raw URL.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 194 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:contact

An individual contact from a domain whois record.

The base type for the form can be found at *inet:whois:contact*.

Properties:

name	type	doc	opts
:rec	*inet:whois:rec*	The whois record containing the contact data.	Read Only: True
:rec:fqdn	*inet:fqdn*	The domain associated with the whois record.	Read Only: True
:rec:asof	*time*	The date of the whois record.	Read Only: True
:type	*str* lower: True	The contact type (e.g., registrar, registrant, admin, billing, tech, etc.).	Read Only: True
:id	*str* lower: True	The ID associated with the contact.	
:name	*str* lower: True	The name of the contact.	
:email	*inet:email*	The email address of the contact.	
:orgname	*ou:name*	The name of the contact organization.	
:address	*str* lower: True	The content of the street address field(s) of the contact.	
:city	*str* lower: True	The content of the city field of the contact.	
:state	*str* lower: True	The content of the state field of the contact.	
:country	*str* lower: True	The two-letter country code of the contact.	
:phone	*tel:phone*	The content of the phone field of the contact.	
:fax	*tel:phone*	The content of the fax field of the contact.	
:url	*inet:url*	The URL specified for the contact.	
:whois:fqdn	*inet:fqdn*	The whois server FQDN for the given contact (most likely a registrar).	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:email

An email address associated with an FQDN via whois registration text.

The base type for the form can be found at *inet:whois:email*.

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain with a whois record containing the email address.	Read Only: True
:email	*inet:email*	The email address associated with the domain whois record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 196 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:ipcontact

An individual contact from an IP block record.

The base type for the form can be found at *inet:whois:ipcontact*.

Properties:

name	type	doc
:contact	*ps:contact*	Contact information associated with a registration.
:asof	*time*	The date of the record.
:created	*time*	The "created" time from the record.
:updated	*time*	The "last updated" time from the record.
:role	*str* lower: True	The primary role for the contact.
:roles	*array* type: *str* uniq: True sorted: True	Additional roles assigned to the contact.
:asn	*inet:asn*	The associated Autonomous System Number (ASN).
:id	*inet:whois:regid*	The registry unique identifier (e.g. NET-74-0-0-0-1).
:links	*array* type: *inet:url* uniq: True sorted: True	URLs provided with the record.
:status	*str* lower: True	The state of the registered contact (e.g. validated, obscured).
:contacts	*array* type: *inet:whois:ipcontact* uniq: True sorted: True	Additional contacts referenced by this contact.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:ipquery

Query details used to retrieve an IP record.

The base type for the form can be found at *inet:whois:ipquery*.

Properties:

name	type	doc
:time	*time*	The time the request was made.
:url	*inet:url*	The query URL when using the HTTP RDAP Protocol.
:fqdn	*inet:fqdn*	The FQDN of the host server when using the legacy WHOIS Protocol.
:ipv4	*inet:ipv4*	The IPv4 address queried.
:ipv6	*inet:ipv6*	The IPv6 address queried.
:success	*bool*	Whether the host returned a valid response for the query.
:rec	*inet:whois:iprec*	The resulting record from the query.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.

continues on next page

Table 198 – continued from previous page

source	verb	target	doc
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:iprec

An IPv4/IPv6 block registration record.

The base type for the form can be found at *inet:whois:iprec*.

Properties:

name	type	doc	opts
`:net4`	*inet:net4*	The IPv4 address range assigned.	
`:net4:min`	*inet:ipv4*	The first IPv4 in the range assigned.	
`:net4:max`	*inet:ipv4*	The last IPv4 in the range assigned.	
`:net6`	*inet:net6*	The IPv6 address range assigned.	
`:net6:min`	*inet:ipv6*	The first IPv6 in the range assigned.	
`:net6:max`	*inet:ipv6*	The last IPv6 in the range assigned.	
`:asof`	*time*	The date of the record.	
`:created`	*time*	The “created” time from the record.	
`:updated`	*time*	The “last updated” time from the record.	
`:text`	*str* lower: `True`	The full text of the record.	Display: `{'hint': 'text'}`
`:desc`	*str* lower: `True`	Notes concerning the record.	Display: `{'hint': 'text'}`
`:asn`	*inet:asn*	The associated Autonomous System Number (ASN).	
`:id`	*inet:whois:regid*	The registry unique identifier (e.g. NET-74-0-0-0-1).	
`:name`	*str*	The name assigned to the network by the registrant.	
`:parentid`	*inet:whois:regid*	The registry unique identifier of the parent whois record (e.g. NET-74-0-0-0-0).	
`:registrant`	*inet:whois:ipcontact*	Deprecated. Add the registrant inet:whois:ipcontact to the :contacts array.	Deprecated: `True`
`:contacts`	*array* type: *inet:whois:ipcontact* uniq: `True` sorted: `True`	Additional contacts from the record.	
`:country`	*str* lower: `True` regex: `^[a-z]{2}$`	The two-letter ISO 3166 country code.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
inet:whois:ip	-(ipwhois	inet:ipv4	The source IP whois record describes the target IPv4 address.
inet:whois:ip	-(ipwhois	inet:ipv6	The source IP whois record describes the target IPv6 address.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:rar

A domain registrar.

The base type for the form can be found at *inet:whois:rar*.

An example of `inet:whois:rar`:

- `godaddy, inc.`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

continues on next

Table 200 – continued from previous page

source	verb	target	doc
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:rec

A domain whois record.

The base type for the form can be found at *inet:whois:rec*.

Properties:

name	type	doc	opts
:fqdn	*inet:fqdn*	The domain associated with the whois record.	Read Only: True
:asof	*time*	The date of the whois record.	Read Only: True
:text	*str* lower: True	The full text of the whois record.	Display: {'hint': 'text'}
:created	*time*	The "created" time from the whois record.	
:updated	*time*	The "last updated" time from the whois record.	
:expires	*time*	The "expires" time from the whois record.	
:registrar	*inet:whois:rar*	The registrar name from the whois record.	
:registrant	*inet:whois:reg*	The registrant name from the whois record.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:recns

A nameserver associated with a domain whois record.

The base type for the form can be found at *inet:whois:recns*.

Properties:

name	type	doc	opts
:ns	*inet:fqdn*	A nameserver for a domain as listed in the domain whois record.	Read Only: True
:rec	*inet:whois:re*	The whois record containing the nameserver data.	Read Only: True
:rec:fqdn	*inet:fqdn*	The domain associated with the whois record.	Read Only: True
:rec:aso:	*time*	The date of the whois record.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:reg

A domain registrant.

The base type for the form can be found at *inet:whois:reg*.

An example of `inet:whois:reg`:

- `woot hostmaster`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

continues on next

Table 203 – continued from previous page

source	verb	target	doc
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:whois:regid

The registry unique identifier of the registration record.

The base type for the form can be found at *inet:whois:regid*.

An example of `inet:whois:regid`:

- `NET-10-0-0-0-1`

  Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next

Table 204 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:wifi:ap

An SSID/MAC address combination for a wireless access point.

The base type for the form can be found at *inet:wifi:ap*.

Properties:

name	type	doc	opts
:ssid	*inet:wifi:ssid*	The SSID for the wireless access point.	Read Only: True
:bssid	*inet:mac*	The MAC address for the wireless access point.	Read Only: True
:latlong	*geo:latlong*	The best known latitude/longitude for the wireless access point.	
:accuracy	*geo:dist*	The reported accuracy of the latlong telemetry reading.	
:channel	*int*	The WIFI channel that the AP was last observed operating on.	
:encryption	*str* lower: True strip: True	The type of encryption used by the WIFI AP such as “wpa2”.	
:place	*geo:place*	The geo:place associated with the latlong property.	
:loc	*loc*	The geo-political location string for the wireless access point.	
:org	*ou:org*	The organization that owns/operates the access point.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

inet:wifi:ssid

A WiFi service set identifier (SSID) name.

The base type for the form can be found at *inet:wifi:ssid*.

An example of `inet:wifi:ssid`:

- `The Vertex Project`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

continues on next

Table 206 – continued from previous page

source	verb	target	doc
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

iso:oid

An ISO Object Identifier string.

The base type for the form can be found at *iso:oid*.

Properties:

name	type	doc
:descr	*str*	A description of the value or meaning of the OID.
:identifier	*str*	The string identifier for the deepest tree element.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 207 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:account

A GUID that represents an account on a host or network.

The base type for the form can be found at *it:account*.

Properties:

name	type	doc	opts
:user	*inet:user*	The username associated with the account.	
:contact	*ps:contact*	Additional contact information associated with this account.	
:host	*it:host*	The host where the account is registered.	
:domain	*it:domain*	The authentication domain where the account is registered.	
:posix:uid	*int*	The user ID of the account.	Example: `1001`
:posix:gid	*int*	The primary group ID of the account.	Example: `1001`
:posix:gecos	*int*	The GECOS field for the POSIX account.	
:posix:home	*file:path*	The path to the POSIX account's home directory.	Example: `/home/visi`
:posix:shell	*file:path*	The path to the POSIX account's default shell.	Example: `/bin/bash`
:windows:sid	*it:os:windows:sid*	The Microsoft Windows Security Identifier of the account.	
:groups	*array* type: *it:group* uniq: `True` sorted: `True`	An array of groups that the account is a member of.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 208 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:adid

An advertising identification string.

The base type for the form can be found at *it:adid*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 209 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:app:snort:hit

An instance of a snort rule hit.

The base type for the form can be found at *it:app:snort:hit*.

Properties:

name	type	doc
:rule	*it:app:snort:rule*	The snort rule that matched the file.
:flow	*inet:flow*	The inet:flow that matched the snort rule.
:src	*inet:addr*	The source address of flow that caused the hit.
:src:ipv4	*inet:ipv4*	The source IPv4 address of the flow that caused the hit.
:src:ipv6	*inet:ipv6*	The source IPv6 address of the flow that caused the hit.
:src:port	*inet:port*	The source port of the flow that caused the hit.
:dst	*inet:addr*	The destination address of the trigger.
:dst:ipv4	*inet:ipv4*	The destination IPv4 address of the flow that caused the hit.
:dst:ipv6	*inet:ipv6*	The destination IPv4 address of the flow that caused the hit.
:dst:port	*inet:port*	The destination port of the flow that caused the hit.
:time	*time*	The time of the network flow that caused the hit.
:sensor	*it:host*	The sensor host node that produced the hit.
:version	*it:semver*	The version of the rule at the time of match.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 210 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:app:snort:rule

A snort rule.

The base type for the form can be found at *it:app:snort:rule*.

Properties:

name	type	doc	opts
:id	*str*	The snort rule id.	
:text	*str*	The snort rule text.	Display: {'hint': 'text'}
:name	*str*	The name of the snort rule.	
:desc	*str*	A brief description of the snort rule.	Display: {'hint': 'text'}
:engine	*int*	The snort engine ID which can parse and evaluate the rule text.	
:versio	*it:semver*	The current version of the rule.	
:author	*ps:contact*	Contact info for the author of the rule.	
:create	*time*	The time the rule was initially created.	
:update	*time*	The time the rule was most recently modified.	
:enable	*bool*	The rule enabled status to be used for snort evaluation engines.	
:family	*it:prod:softna*	The name of the software family the rule is designed to detect.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat):	geo:telem	The source node was seen at the geo:telem node place and time.
it:app:snort:r	-(detects)	*	The snort rule is intended for use in detecting the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:app:yara:match

A YARA rule match to a file.

The base type for the form can be found at *it:app:yara:match*.

Properties:

name	type	doc	opts
:rule	*it:app:yara:rule*	The YARA rule that matched the file.	Read Only: True
:file	*file:bytes*	The file that matched the YARA rule.	Read Only: True
:version	*it:semver*	The most recent version of the rule evaluated as a match.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:app:yara:procmatch

An instance of a YARA rule match to a process.

The base type for the form can be found at *it:app:yara:procmatch*.

Properties:

name	type	doc
:rule	*it:app:yara:rule*	The YARA rule that matched the file.
:proc	*it:exec:proc*	The process that matched the YARA rule.
:time	*time*	The time that the YARA engine matched the process to the rule.
:version	*it:semver*	The most recent version of the rule evaluated as a match.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 213 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:app:yara:rule

A YARA rule unique identifier.

The base type for the form can be found at *it:app:yara:rule*.

Properties:

name	type	doc	opts
:text	*str*	The YARA rule text.	Display: {'hint': 'text', 'syntax': 'yara'}
:ext:i	*str*	The YARA rule ID from an external system.	
:url	*inet:url*	A URL which documents the YARA rule.	
:name	*str*	The name of the YARA rule.	
:autho	*ps:contact*	Contact info for the author of the YARA rule.	
:versi	*it:semver*	The current version of the rule.	
:creat	*time*	The time the YARA rule was initially created.	
:updat	*time*	The time the YARA rule was most recently modified.	
:enabl	*bool*	The rule enabled status to be used for YARA evaluation engines.	
:famil	*it:prod:soft*	The name of the software family the rule is designed to detect.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
it:app:yara:r	-(detects)	*	The YARA rule is intended for use in detecting the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:auth:passwdhash

An instance of a password hash.

The base type for the form can be found at *it:auth:passwdhash*.

Properties:

name	type	doc
:salt	*hex*	The (optional) hex encoded salt value used to calculate the password hash.
:hash:md5	*hash:md5*	The MD5 password hash value.
:hash:sha1	*hash:sha1*	The SHA1 password hash value.
:hash:sha256	*hash:sha256*	The SHA256 password hash value.
:hash:sha512	*hash:sha512*	The SHA512 password hash value.
:hash:lm	*hash:lm*	The LM password hash value.
:hash:ntlm	*hash:ntlm*	The NTLM password hash value.
:passwd	*inet:passwd*	The (optional) clear text password for this password hash.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:av:filehit

Deprecated. Please use it:av:scan:result.

The base type for the form can be found at *it:av:filehit*.

Properties:

name	type	doc	opts
:file	*file:bytes*	The file that triggered the signature hit.	Read Only: True
:sig	*it:av:sig*	The signature that the file triggered on.	Read Only: True
:sig:name	*it:av:signame*	The signature name.	Read Only: True
:sig:soft	*it:prod:soft*	The anti-virus product which contains the signature.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 216 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:av:prochit

Deprecated. Please use it:av:scan:result.

The base type for the form can be found at *it:av:prochit*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The file that triggered the signature hit.
:sig	*it:av:sig*	The signature that the file triggered on.
:time	*time*	The time that the AV engine detected the signature.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 217 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:av:scan:result

The result of running an antivirus scanner.

The base type for the form can be found at *it:av:scan:result*.

Properties:

name	type	doc
:time	*time*	The time the scan was run.
:verdict	*int* enums: ((10, 'benign'), (20, 'unknown'), (30, 'suspicious'), (40, 'malicious'))	The scanner provided verdict for the scan.
:scanner	*it:prod:softver*	The scanner software used to produce the result.
:scanner:name	*it:prod:softname*	The name of the scanner software.
:signame	*it:av:signame*	The name of the signature returned by the scanner.
:target:file	*file:bytes*	The file that was scanned to produce the result.
:target:proc	*it:exec:proc*	The process that was scanned to produce the result.
:target:host	*it:host*	The host that was scanned to produce the result.
:target:fqdn	*inet:fqdn*	The FQDN that was scanned to produce the result.
:target:url	*inet:url*	The URL that was scanned to produce the result.
:target:ipv4	*inet:ipv4*	The IPv4 address that was scanned to produce the result.
:target:ipv6	*inet:ipv6*	The IPv6 address that was scanned to produce the result.
:multi:scan	*it:av:scan:result*	Set if this result was part of running multiple scanners.
:multi:count	*int* min: 0	The total number of scanners which were run by a multi-scanner.
:multi:count:benign	*int* min: 0	The number of scanners which returned a benign verdict.
:multi:count:unknown	*int* min: 0	The number of scanners which returned a unknown/unsupported verdict.
:multi:count:suspicious	*int* min: 0	The number of scanners which returned a suspicious verdict.
:multi:count:malicious	*int* min: 0	The number of scanners which returned a malicious verdict.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:av:sig

Deprecated. Please use it:av:scan:result.

The base type for the form can be found at *it:av:sig*.

Properties:

name	type	doc	opts
:soft	*it:prod:soft*	The anti-virus product which contains the signature.	Read Only: True
:name	*it:av:signame*	The signature name.	Read Only: True
:desc	*str*	A free-form description of the signature.	Display: {'hint': 'text'}
:url	*inet:url*	A reference URL for information about the signature.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 219 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:av:signame

An antivirus signature name.

The base type for the form can be found at *it:av:signame*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 220 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:cmd

A unique command-line string.

The base type for the form can be found at *it:cmd*.

An example of `it:cmd`:

- `foo.exe --dostuff bar`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next

Table 221 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:int

A developer selected integer constant.

The base type for the form can be found at *it:dev:int*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 222 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:mutex

A string representing a mutex.

The base type for the form can be found at *it:dev:mutex*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 223 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:pipe

A string representing a named pipe.

The base type for the form can be found at *it:dev:pipe*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 224 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:regkey

A Windows registry key.

The base type for the form can be found at *it:dev:regkey*.

An example of `it:dev:regkey`:

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next

Table 225 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:regval

A Windows registry key/value pair.

The base type for the form can be found at *it:dev:regval*.

Properties:

name	type	doc
:key	*it:dev:regkey*	The Windows registry key.
:str	*it:dev:str*	The value of the registry key, if the value is a string.
:int	*it:dev:int*	The value of the registry key, if the value is an integer.
:bytes	*file:bytes*	The file representing the value of the registry key, if the value is binary data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo

A version control system instance.

The base type for the form can be found at *it:dev:repo*.

Properties:

name	type	doc	opts
:name	*str* lower: `True` strip: `True`	The name of the repository.	
:desc	*str*	A free-form description of the repository.	Display: `{'hint': 'text'}`
:created	*time*	When the repository was created.	
:url	*inet:url*	A URL where the repository is hosted.	
:type	*it:dev:repo:type:taxono*	The type of the version control system used.	Example: `svn`
:submodules	*array* type: *it:dev:repo:commit*	An array of other repos that this repo has as submodules, pinned at specific commits.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 227 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:branch

A branch in a version control system instance.

The base type for the form can be found at *it:dev:repo:branch*.

Properties:

name	type	doc
:parent	*it:dev:repo:branch*	The branch this branch was branched from.
:start	*it:dev:repo:commit*	The commit in the parent branch this branch was created at.
:name	*str* strip: True	The name of the branch.
:url	*inet:url*	The URL where the branch is hosted.
:created	*time*	The time this branch was created.
:merged	*time*	The time this branch was merged back into its parent.
:deleted	*time*	The time this branch was deleted.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:commit

A commit to a repository.

The base type for the form can be found at *it:dev:repo:commit*.

Properties:

name	type	doc	opts
:repo	*it:dev:repo*	The repository the commit lives in.	
:parents	*array* type: *it:dev:repo:commit*	The commit or commits this commit is immediately based on.	
:branch	*it:dev:repo:branch*	The name of the branch the commit was made to.	
:mesg	*str*	The commit message describing the changes in the commit.	Display: {'hint': 'text'}
:id	*str*	The version control system specific commit identifier.	
:created	*time*	The time the commit was made.	
:url	*inet:url*	The URL where the commit is hosted.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next page

Table 229 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:diff

A diff of a file being applied in a single commit.

The base type for the form can be found at *it:dev:repo:diff* .

Properties:

name	type	doc
:commit	*it:dev:repo:commit*	The commit that produced this diff.
:file	*file:bytes*	The file after the commit has been applied.
:path	*file:path*	The path to the file in the repo that the diff is being applied to.
:url	*inet:url*	The URL where the diff is hosted.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:diff:comment

A comment on a diff in a repository.

The base type for the form can be found at *it:dev:repo:diff:comment*.

Properties:

name	type	doc	opts
:diff	*it:dev:repo:diff*	The diff the comment is being added to.	
:text	*str*	The body of the comment.	Display: {'hint': 'text'}
:replyt	*it:dev:repo:diff:co*	The comment that this comment is replying to.	
:line	*int*	The line in the file that is being commented on.	
:offset	*int*	The offset in the line in the file that is being commented on.	
:url	*inet:url*	The URL where the comment is hosted.	
:create	*time*	The time the comment was created.	
:update	*time*	The time the comment was updated.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 231 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:issue

An issue raised in a repository.

The base type for the form can be found at *it:dev:repo:issue*.

Properties:

name	type	doc	opts
:repo	*it:dev:repo*	The repo where the issue was logged.	
:title	*str* lower: True strip: True	The title of the issue.	
:desc	*str*	The text describing the issue.	Display: {'hint': 'text'}
:created	*time*	The time the issue was created.	
:updated	*time*	The time the issue was updated.	
:url	*inet:url*	The URL where the issue is hosted.	
:id	*str* strip: True	The ID of the issue in the repository system.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:issue:comment

A comment on an issue in a repository.

The base type for the form can be found at *it:dev:repo:issue:comment*.

Properties:

name	type	doc	opts
:issue	*it:dev:repo:issue*	The issue thread that the comment was made in.	
:text	*str*	The body of the comment.	Display: {'hint': 'text'}
:replyt	*it:dev:repo:issue:com*	The comment that this comment is replying to.	
:url	*inet:url*	The URL where the comment is hosted.	
:create	*time*	The time the comment was created.	
:update	*time*	The time the comment was updated.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:issue:label

A label applied to a repository issue.

The base type for the form can be found at *it:dev:repo:issue:label*.

Properties:

name	type	doc
:issue	*it:dev:repo:issue*	The issue the label was applied to.
:label	*it:dev:repo:label*	The label that was applied to the issue.
:applied	*time*	The time the label was applied.
:removed	*time*	The time the label was removed.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 234 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:label

A developer selected label.

The base type for the form can be found at *it:dev:repo:label*.

Properties:

name	type	doc	opts
:id	*str* strip: True	The ID of the label.	
:title	*str* lower: True strip: True	The human friendly name of the label.	
:desc	*str*	The description of the label.	Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 235 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:remote

A remote repo that is tracked for changes/branches/etc.

The base type for the form can be found at *it:dev:repo:remote*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name the repo is using for the remote repo.	Example: `origin`
:url	*inet:url*	The URL the repo is using to access the remote repo.	
:repo	*it:dev:repo*	The repo that is tracking the remote repo.	
:remote	*it:dev:repo*	The instance of the remote repo.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:repo:type:taxonomy

A version control system type taxonomy.

The base type for the form can be found at *it:dev:repo:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*it:dev:repo:type:taxono*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 237 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:dev:str

A developer selected string.

The base type for the form can be found at *it:dev:str*.

Properties:

name	type	doc
:norm	*str* lower: True	Lower case normalized version of the it:dev:str.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 238 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:domain

A logical boundary of authentication and configuration such as a windows domain.

The base type for the form can be found at *it:domain*.

Properties:

name	type	doc
:name	*str* lower: True onespace: True	The name of the domain.
:desc	*str*	A brief description of the domain.
:org	*ou:org*	The org that operates the given domain.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:bind

An instance of a host binding a listening port.

The base type for the form can be found at *it:exec:bind*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:server	*inet:server*	The inet:addr of the server when binding the port.
:server:ipv4	*inet:ipv4*	The IPv4 address specified to bind().
:server:ipv6	*inet:ipv6*	The IPv6 address specified to bind().
:server:port	*inet:port*	The bound (listening) TCP port.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 240 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:file:add

An instance of a host adding a file to a filesystem.

The base type for the form can be found at *it:exec:file:add*.

Properties:

name	type	doc	opts
:proc	*it:exec:proc*	The host process which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:exe	*file:bytes*	The executable file which caused the activity.	
:time	*time*	The time that the activity started.	
:path	*file:path*	The path where the file was created.	
:path:dir	*file:path*	The parent directory of the file path (parsed from :path).	Read Only: True
:path:ext	*str* lower: True strip: True	The file extension of the file name (parsed from :path).	Read Only: True
:path:base	*file:base*	The final component of the file path (parsed from :path).	Read Only: True
:file	*file:bytes*	The file that was created.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:thread	*it:exec:thread*	The host thread which caused the activity.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 241 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:file:del

An instance of a host deleting a file from a filesystem.

The base type for the form can be found at *it:exec:file:del*.

Properties:

name	type	doc	opts
:proc	*it:exec:proc*	The host process which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:exe	*file:bytes*	The executable file which caused the activity.	
:time	*time*	The time that the activity started.	
:path	*file:path*	The path where the file was deleted.	
:path:dir	*file:path*	The parent directory of the file path (parsed from :path).	Read Only: True
:path:ext	*str* lower: True strip: True	The file extension of the file name (parsed from :path).	Read Only: True
:path:base	*file:base*	The final component of the file path (parsed from :path).	Read Only: True
:file	*file:bytes*	The file that was deleted.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:thread	*it:exec:thread*	The host thread which caused the activity.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 242 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:file:read

An instance of a host reading a file from a filesystem.

The base type for the form can be found at *it:exec:file:read*.

Properties:

name	type	doc	opts
:proc	*it:exec:proc*	The host process which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:exe	*file:bytes*	The executable file which caused the activity.	
:time	*time*	The time that the activity started.	
:path	*file:path*	The path where the file was read.	
:path:dir	*file:path*	The parent directory of the file path (parsed from :path).	Read Only: `True`
:path:ext	*str* lower: `True` strip: `True`	The file extension of the file name (parsed from :path).	Read Only: `True`
:path:base	*file:base*	The final component of the file path (parsed from :path).	Read Only: `True`
:file	*file:bytes*	The file that was read.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:thread	*it:exec:thread*	The host thread which caused the activity.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.

continues on next pag

Table 243 – continued from previous page

source	verb	target	doc
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:file:write

An instance of a host writing a file to a filesystem.

The base type for the form can be found at *it:exec:file:write*.

Properties:

name	type	doc	opts
:proc	*it:exec:proc*	The host process which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:exe	*file:bytes*	The executable file which caused the activity.	
:time	*time*	The time that the activity started.	
:path	*file:path*	The path where the file was written to/modified.	
:path:dir	*file:path*	The parent directory of the file path (parsed from :path).	Read Only: True
:path:ext	*str* lower: True strip: True	The file extension of the file name (parsed from :path).	Read Only: True
:path:base	*file:base*	The final component of the file path (parsed from :path).	Read Only: True
:file	*file:bytes*	The file that was modified.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:thread	*it:exec:thread*	The host thread which caused the activity.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 244 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:loadlib

A library load event in a process.

The base type for the form can be found at *it:exec:loadlib*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:va	*int*	The base memory address where the library was loaded in the process.
:loaded	*time*	The time the library was loaded.
:unloaded	*time*	The time the library was unloaded.
:path	*file:path*	The path that the library was loaded from.
:file	*file:bytes*	The library file that was loaded.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:exe	*file:bytes*	The executable file which caused the activity.
:thread	*it:exec:thread*	The host thread which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:time	*time*	The time that the activity started.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:mmap

A memory mapped segment located in a process.

The base type for the form can be found at *it:exec:mmap*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:va	*int*	The base memory address where the map was created in the process.
:size	*int*	The size of the memory map in bytes.
:perms:read	*bool*	True if the mmap is mapped with read permissions.
:perms:write	*bool*	True if the mmap is mapped with write permissions.
:perms:execu	*bool*	True if the mmap is mapped with execute permissions.
:created	*time*	The time the memory map was created.
:deleted	*time*	The time the memory map was deleted.
:path	*file:path*	The file path if the mmap is a mapped view of a file.
:hash:sha256	*hash:sha256*	A SHA256 hash of the memory map. Bytes may optionally be present in the axon.
:sandbox:fil	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:exe	*file:bytes*	The executable file which caused the activity.
:thread	*it:exec:threa*	The host thread which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:time	*time*	The time that the activity started.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 246 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:mutex

A mutex created by a process at runtime.

The base type for the form can be found at *it:exec:mutex*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:name	*it:dev:mutex*	The mutex string.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:pipe

A named pipe created by a process at runtime.

The base type for the form can be found at *it:exec:pipe*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:name	*it:dev:pipe*	The named pipe string.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:proc

A process executing on a host. May be an actual (e.g., endpoint) or virtual (e.g., malware sandbox) host.

The base type for the form can be found at *it:exec:proc*.

Properties:

name	type	doc	opts
:host	*it:host*	The host on which the activity occurred.	
:exe	*file:bytes*	The executable file which caused the activity.	
:cmd	*it:cmd*	The command string used to launch the process, including any command line parameters.	Display: {'hint': 'text'}
:pid	*int*	The process ID.	
:time	*time*	The time that the activity started.	
:name	*str*	The display name specified by the process.	
:exited	*time*	The time the process exited.	
:exitcode	*int*	The exit code for the process.	
:user	*inet:user*	The user name of the process owner.	Deprecated: True
:account	*it:accoun*	The account of the process owner.	
:path	*file:path*	The path to the executable of the process.	
:path:bas	*file:base*	The file basename of the executable of the process.	
:src:exe	*file:path*	Deprecated. Create :src:proc and set :path.	Deprecated: True
:src:proc	*it:exec:pr*	The process which created the process.	
:killedby	*it:exec:pr*	The process which killed this process.	
:sandbox:	*file:bytes*	The initial sample given to a sandbox environment to analyze.	
:proc	*it:exec:pr*	The host process which caused the activity.	
:thread	*it:exec:th*	The host thread which caused the activity.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next page

Table 249 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:query

An instance of an executed query.

The base type for the form can be found at *it:exec:query*.

Properties:

name	type	doc
:text	*it:query*	The query string that was executed.
:opts	*data*	An opaque JSON object containing query parameters and options.
:api:url	*inet:url*	The URL of the API endpoint the query was sent to.
:language	*str* lower: True onespace: True	The name of the language that the query is expressed in.
:offset	*int*	The offset of the last record consumed from the query.
:exe	*file:bytes*	The executable file which caused the activity.
:proc	*it:exec:proc*	The host process which caused the activity.
:thread	*it:exec:thread*	The host thread which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:time	*time*	The time that the activity started.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
it:exec:que	-(found)>	*	The target node was returned as a result of running the query.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 250 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:reg:del

An instance of a host deleting a registry key.

The base type for the form can be found at *it:exec:reg:del*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:reg	*it:dev:regval*	The registry key or value that was deleted.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:reg:get

An instance of a host getting a registry key.

The base type for the form can be found at *it:exec:reg:get*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:reg	*it:dev:regval*	The registry key or value that was read.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 252 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:reg:set

An instance of a host creating or setting a registry key.

The base type for the form can be found at *it:exec:reg:set*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:reg	*it:dev:regval*	The registry key or value that was written to.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next page

Table 253 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:thread

A thread executing in a process.

The base type for the form can be found at *it:exec:thread*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:created	*time*	The time the thread was created.
:exited	*time*	The time the thread exited.
:exitcode	*int*	The exit code or return value for the thread.
:src:proc	*it:exec:proc*	An external process which created the thread.
:src:thread	*it:exec:thread*	The thread which created this thread.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:exe	*file:bytes*	The executable file which caused the activity.
:thread	*it:exec:thread*	The host thread which caused the activity.
:host	*it:host*	The host on which the activity occurred.
:time	*time*	The time that the activity started.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:exec:url

An instance of a host requesting a URL.

The base type for the form can be found at *it:exec:url*.

Properties:

name	type	doc
:proc	*it:exec:proc*	The host process which caused the activity.
:browser	*it:prod:softver*	The software version of the browser.
:host	*it:host*	The host on which the activity occurred.
:exe	*file:bytes*	The executable file which caused the activity.
:time	*time*	The time that the activity started.
:url	*inet:url*	The URL that was requested.
:page:pdf	*file:bytes*	The rendered DOM saved as a PDF file.
:page:html	*file:bytes*	The rendered DOM saved as an HTML file.
:page:image	*file:bytes*	The rendered DOM saved as an image.
:http:request	*inet:http:request*	The HTTP request made to retrieve the initial URL contents.
:client	*inet:client*	The address of the client during the URL retrieval.
:client:ipv4	*inet:ipv4*	The IPv4 of the client during the URL retrieval.
:client:ipv6	*inet:ipv6*	The IPv6 of the client during the URL retrieval.
:client:port	*inet:port*	The client port during the URL retrieval.
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.
:thread	*it:exec:thread*	The host thread which caused the activity.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 255 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:fs:file

A file on a host.

The base type for the form can be found at *it:fs:file*.

Properties:

name	type	doc	opts
:host	*it:host*	The host containing the file.	
:path	*file:path*	The path for the file.	
:path:dir	*file:path*	The parent directory of the file path (parsed from :path).	Read Only: True
:path:ext	*str* lower: True strip: True	The file extension of the file name (parsed from :path).	Read Only: True
:path:base	*file:base*	The final component of the file path (parsed from :path).	Read Only: True
:file	*file:bytes*	The file on the host.	
:ctime	*time*	The file creation time.	
:mtime	*time*	The file modification time.	
:atime	*time*	The file access time.	
:user	*inet:user*	The owner of the file.	
:group	*inet:user*	The group owner of the file.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 256 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:group

A GUID that represents a group on a host or network.

The base type for the form can be found at *it:group*.

Properties:

name	type	doc	opts
:name	*str* lower: `True` onespace: `True`	The name of the group.	
:desc	*str*	A brief description of the group.	
:host	*it:host*	The host where the group is registered.	
:domain	*it:domain*	The authentication domain where the group is registered.	
:groups	*array* type: *it:group* uniq: `True` sorted: `True`	Groups that are a member of this group.	
:posix:gid	*int*	The primary group ID of the account.	Example: `1001`
:windows:sid	*it:os:windows:sid*	The Microsoft Windows Security Identifier of the group.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 257 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:host

A GUID that represents a host or system.

The base type for the form can be found at *it:host*.

Properties:

name	type	doc	opts
:name	*it:hostname*	The name of the host or system.	
:desc	*str*	A free-form description of the host.	
:domain	*it:domain*	The authentication domain that the host is a member of.	
:ipv4	*inet:ipv4*	The last known ipv4 address for the host.	
:latlong	*geo:latlong*	The last known location for the host.	
:place	*geo:place*	The place where the host resides.	
:loc	*loc*	The geo-political location string for the node.	
:os	*it:prod:softver*	The operating system of the host.	
:os:name	*it:prod:softname*	A software product name for the host operating system. Used for entity resolution.	
:hardware	*it:prod:hardware*	The hardware specification for this host.	
:manu	*str*	Please use :hardware:make.	Deprecated: `True`
:model	*str*	Please use :hardware:model.	Deprecated: `True`
:serial	*str*	The serial number of the host.	
:operator	*ps:contact*	The operator of the host.	
:org	*ou:org*	The org that operates the given host.	
:ext:id	*str*	An external identifier for the host.	
:keyboard:layout	*str* lower: `True` onespace: `True`	The primary keyboard layout configured on the host.	
:keyboard:language	*lang:language*	The primary keyboard input language configured on the host.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:hostname

The name of a host or system.

The base type for the form can be found at *it:hostname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:hostsoft

A version of a software product which is present on a given host.

The base type for the form can be found at *it:hostsoft*.

Properties:

name	type	doc	opts
:host	*it:host*	Host with the software.	Read Only: True
:softver	*it:prod:softver*	Software on the host.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 260 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:hosturl

A url hosted on or served by a host or system.

The base type for the form can be found at *it:hosturl*.

Properties:

name	type	doc	opts
:host	*it:host*	Host serving a url.	Read Only: True
:url	*inet:url*	URL available on the host.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 261 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:log:event

A GUID representing an individual log event.

The base type for the form can be found at *it:log:event*.

Properties:

name	type	doc	opts
:mesg	*str*	The log message text.	
:type	*it:log:event:type:taxono*	A taxonometric type for the log event.	Example: `windows.eventlog.securitylog`
:severity	*int* enums: `((10, 'debug'), (20, 'info'), (30, 'notice'), (40, 'warning'), (50, 'err'), (60, 'crit'), (70, 'alert'), (80, 'emerg'))`	A log level integer that increases with severity.	
:data	*data*	A raw JSON record of the log event.	
:ext:id	*str*	An external id that uniquely identifies this log entry.	
:product	*it:prod:softver*	The software which produced the log entry.	
:exe	*file:bytes*	The executable file which caused the activity.	
:proc	*it:exec:proc*	The host process which caused the activity.	
:thread	*it:exec:thread*	The host thread which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:time	*time*	The time that the activity started.	
:sandbox:file	*file:bytes*	The initial sample given to a sandbox environment to analyze.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:log:event:type:taxonomy

A taxonomy of log event types.

The base type for the form can be found at *it:log:event:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*it:log:event:type:taxonc*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 263 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:logon

A GUID that represents an individual logon/logoff event.

The base type for the form can be found at *it:logon*.

Properties:

name	type	doc
:time	*time*	The time the logon occurred.
:success	*bool*	Set to false to indicate an unsuccessful logon attempt.
:logoff:time	*time*	The time the logon session ended.
:host	*it:host*	The host that the account logged in to.
:account	*it:account*	The account that logged in.
:creds	*auth:creds*	The credentials that were used for the logon.
:duration	*duration*	The duration of the logon session.
:client:host	*it:host*	The host where the logon originated.
:client:ipv4	*inet:ipv4*	The IPv4 where the logon originated.
:client:ipv6	*inet:ipv6*	The IPv6 where the logon originated.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 264 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:campaign

A MITRE ATT&CK Campaign ID.

The base type for the form can be found at *it:mitre:attack:campaign*.

An example of `it:mitre:attack:campaign`:

- `C0028`

Properties:

name	type	doc	opts
:name	*ou:campname*	The primary name for the ATT&CK campaign.	
:names	*array* type: *ou:campname* uniq: True sorted: True	An array of alternate names for the ATT&CK campaign.	
:desc	*str* strip: True	A description of the ATT&CK campaign.	Display: {'hint': 'text'}
:url	*inet:url*	The URL that documents the ATT&CK campaign.	
:groups	*array* type: *it:mitre:attack:group* uniq: True sorted: True split: ,	An array of ATT&CK group IDs attributed to the campaign.	
:software	*array* type: *it:mitre:attack:softwa* uniq: True sorted: True split: ,	An array of ATT&CK software IDs used in the campaign.	
:techniques	*array* type: *it:mitre:attack:techni* uniq: True sorted: True split: ,	An array of ATT&CK technique IDs used in the campaign.	
:matrices	*array* type: *it:mitre:attack:matri* uniq: True sorted: True split: ,	The ATT&CK matrices which define the campaign.	
:references	*array*	An array of URLs that document the	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:flow

A MITRE ATT&CK Flow diagram.

The base type for the form can be found at *it:mitre:attack:flow*.

Properties:

name	type	doc
:name	*str*	The name of the attack-flow diagram.
:data	*data* schema: {'$schema': 'https://json-schema.org/draft/2020-12/schema', '$id': 'https://center-for-threat-inform github.io/attack-flow/schema/attack-flow-schema-2.0.0.json', 'title': 'Attack Flow STIX 2.1 Extension', 'description': 'This schema is the normative definition of the STIX 2.1 extension `extension-definition--f 24172197f4`. It extends STIX with additional STIX Data Objects (SDOs) that model Attack Flow concepts.', 'type': 'object', 'unevaluatedProperties': False, 'allOf': ({'$ref': 'http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/common/core.json'},), 'properties': {'type': {'const': 'bundle'}, 'objects': {'type': 'array', 'items': {'$comment': 'Try each of the Attack Flow types in order, eventually falling through to the STIX common object definition.', 'if': {'type': 'object', 'properties': {'type': {'type': 'string', 'const': 'attack-flow'}}}, 'then': {'$ref': 'https://center-for-threat-inform github.io/attack-flow/schema/	The ATT&CK Flow diagram. Schema version 2.0.0 enforced.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:group

A MITRE ATT&CK Group ID.

The base type for the form can be found at *it:mitre:attack:group*.

An example of `it:mitre:attack:group`:

- `G0100`

  Properties:

name	type	doc	opts
:org	*ou:org*	Used to map an ATT&CK group to a synapse ou:org.	
:name	*ou:name*	The primary name for the ATT&CK group.	
:names	*array* type: *ou:name* uniq: `True` sorted: `True`	An array of alternate names for the ATT&CK group.	
:desc	*str*	A description of the ATT&CK group.	Display: `{'hint': 'text'}`
:isnow	*it:mitre:attack:group*	If deprecated, this field may contain the current value for the group.	
:url	*inet:url*	The URL that documents the ATT&CK group.	
:tag	*syn:tag*	Deprecated. Please use a risk:threat:tag.	Deprecated: `True`
:references	*array* type: *inet:url* uniq: `True`	An array of URLs that document the ATT&CK group.	
:techniques	*array* type: *it:mitre:attack:techn* uniq: `True` sorted: `True` split: `,`	An array of ATT&CK technique IDs used by the group.	
:software	*array* type: *it:mitre:attack:softwa* uniq: `True` sorted: `True` split: `,`	An array of ATT&CK software IDs used by the group.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:mitigation

A MITRE ATT&CK Mitigation ID.

The base type for the form can be found at *it:mitre:attack:mitigation*.

An example of `it:mitre:attack:mitigation`:

- `M1036`

Properties:

name	type	doc	opts
`:name`	*str* lower: `True` onespace: `True`	The primary name for the ATT&CK mitigation.	
`:matrix`	*it:mitre:attack:matri*	The ATT&CK matrix which defines the mitigation.	
`:desc`	*str* strip: `True`	A description of the ATT&CK mitigation.	Display: `{'hint': 'text'}`
`:url`	*inet:url*	The URL that documents the ATT&CK mitigation.	
`:tag`	*syn:tag*	Deprecated. Please use risk:mitigation:tag.	Deprecated: `True`
`:references`	*array* type: *inet:url* uniq: `True`	An array of URLs that document the ATT&CK mitigation.	
`:addresses`	*array* type: *it:mitre:attack:techn* uniq: `True` sorted: `True` split: `,`	An array of ATT&CK technique IDs addressed by the mitigation.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:software

A MITRE ATT&CK Software ID.

The base type for the form can be found at *it:mitre:attack:software*.

An example of `it:mitre:attack:software`:

- `S0154`

  Properties:

name	type	doc	opts
:software	*it:prod:soft*	Used to map an ATT&CK software to a synapse it:prod:soft.	
:name	*it:prod:softname*	The primary name for the ATT&CK software.	
:names	*array* type: *it:prod:softname* uniq: `True` sorted: `True`	Associated names for the ATT&CK software.	
:desc	*str* strip: `True`	A description of the ATT&CK software.	Display: `{'hint': 'text'}`
:isnow	*it:mitre:attack:softw*	If deprecated, this field may contain the current value for the software.	
:url	*inet:url*	The URL that documents the ATT&CK software.	
:tag	*syn:tag*	Deprecated. Please use risk:tool:software:tag	Deprecated: `True`
:references	*array* type: *inet:url* uniq: `True`	An array of URLs that document the ATT&CK software.	
:techniques	*array* type: *it:mitre:attack:techni* uniq: `True` sorted: `True` split: `,`	An array of techniques used by the software.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:tactic

A MITRE ATT&CK Tactic ID.

The base type for the form can be found at *it:mitre:attack:tactic*.

An example of `it:mitre:attack:tactic`:

- `TA0040`

Properties:

name	type	doc	opts
`:name`	*str* strip: `True`	The primary name for the ATT&CK tactic.	
`:matrix`	*it:mitre:attack:matri*	The ATT&CK matrix which defines the tactic.	
`:desc`	*str*	A description of the ATT&CK tactic.	Display: `{'hint': 'text'}`
`:url`	*inet:url*	The URL that documents the ATT&CK tactic.	
`:tag`	*syn:tag*	Deprecated.	Deprecated: `True`
`:references`	*array* type: *inet:url* uniq: `True`	An array of URLs that document the ATT&CK tactic.	

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.

continues on next

Table 270 – continued from previous page

source	verb	target	doc
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:mitre:attack:technique

A MITRE ATT&CK Technique ID.

The base type for the form can be found at *it:mitre:attack:technique*.

An example of `it:mitre:attack:technique`:

- `T1548`

  Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The primary name for the ATT&CK technique.	
:matrix	*it:mitre:attack:matri:*	The ATT&CK matrix which defines the technique.	
:status	*it:mitre:attack:status*	The status of this ATT&CK technique.	
:isnow	*it:mitre:attack:techni*	If deprecated, this field may contain the current value for the technique.	
:desc	*str* strip: True	A description of the ATT&CK technique.	Display: {'hint': 'text'}
:url	*inet:url*	The URL that documents the ATT&CK technique.	
:tag	*syn:tag*	Deprecated. Please use ou:technique:tag.	Deprecated: True
:references	*array* type: *inet:url* uniq: True	An array of URLs that document the ATT&CK technique.	
:parent	*it:mitre:attack:techni*	The parent ATT&CK technique on this sub-technique.	
:tactics	*array* type: *it:mitre:attack:tactic* uniq: True sorted: True split: ,	An array of ATT&CK tactics that include this technique.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:network

A GUID that represents a logical network.

The base type for the form can be found at *it:network*.

Properties:

name	type	doc
:name	*str* lower: True onespace: True	The name of the network.
:desc	*str*	A brief description of the network.
:org	*ou:org*	The org that owns/operates the network.
:net4	*inet:net4*	The optional contiguous IPv4 address range of this network.
:net6	*inet:net6*	The optional contiguous IPv6 address range of this network.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 272 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:aaid

An android advertising identification string.

The base type for the form can be found at *it:os:android:aaid*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 273 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:ibroadcast

The given software broadcasts the given Android intent.

The base type for the form can be found at *it:os:android:ibroadcast*.

Properties:

name	type	doc	opts
:app	*it:prod:softver*	The app software which broadcasts the android intent.	Read Only: True
:intent	*it:os:android:inten*	The android intent which is broadcast by the app.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 274 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:ilisten

The given software listens for an android intent.

The base type for the form can be found at *it:os:android:ilisten*.

Properties:

name	type	doc	opts
:app	*it:prod:softver*	The app software which listens for the android intent.	Read Only: True
:intent	*it:os:android:inten*	The android intent which is listened for by the app.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:intent

An android intent string.

The base type for the form can be found at *it:os:android:intent*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:perm

An android permission string.

The base type for the form can be found at *it:os:android:perm*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:android:reqperm

The given software requests the android permission.

The base type for the form can be found at *it:os:android:reqperm*.

Properties:

name	type	doc	opts
:app	*it:prod:softver*	The android app which requests the permission.	Read Only: True
:perm	*it:os:android:perm*	The android permission requested by the app.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.

continues on next page

Table 278 – continued from previous page

source	verb	target	doc
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:os:ios:idfa

An iOS advertising identification string.

The base type for the form can be found at *it:os:ios:idfa*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 279 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:component

A specific instance of an it:prod:hardware most often as part of an it:host.

The base type for the form can be found at *it:prod:component*.

Properties:

name	type	doc
:hardware	*it:prod:hardware*	The hardware specification of this component.
:serial	*str*	The serial number of this component.
:host	*it:host*	The it:host which has this component installed.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 280 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:hardware

A specification for a piece of IT hardware.

The base type for the form can be found at *it:prod:hardware*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The display name for this hardware specification.	
:type	*it:prod:hardwaretype*	The type of hardware.	
:desc	*str*	A brief description of the hardware.	Display: {'hint': 'text'}
:cpe	*it:sec:cpe*	The NIST CPE 2.3 string specifying this hardware.	
:make	*ou:name*	The name of the organization which manufactures this hardware.	
:model	*str* lower: True onespace: True	The model name or number for this hardware specification.	
:version	*str* lower: True onespace: True	Version string associated with this hardware specification.	
:released	*time*	The initial release date for this hardware.	
:parts	*array* type: *it:prod:hardware* uniq: True sorted: True	An array of it:prod:hardware parts included in this hardware specification.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:hardwaretype

An IT hardware type taxonomy.

The base type for the form can be found at *it:prod:hardwaretype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*it:prod:hardwaretype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 282 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:soft

A software product.

The base type for the form can be found at *it:prod:soft*.

Properties:

name	type	doc	opts
:name	*it:prod:softname*	Name of the software.	
:type	*it:prod:soft:taxonomy*	The software type.	
:names	*array* type: *it:prod:softname* uniq: True sorted: True	Observed/variant names for this software.	
:desc	*str*	A description of the software.	Display: {'hint': 'text'}
:desc:short	*str* lower: True	A short description of the software.	
:cpe	*it:sec:cpe*	The NIST CPE 2.3 string specifying this software.	
:author	*ps:contact*	The contact information of the org or person who authored the software.	
:author:org	*ou:org*	Deprecated. Please use :author to link to a ps:contact.	Deprecated: True
:author:acct	*inet:web:acct*	Deprecated. Please use :author to link to a ps:contact.	Deprecated: True
:author:email	*inet:email*	Deprecated. Please use :author to link to a ps:contact.	Deprecated: True
:author:person	*ps:person*	Deprecated. Please use :author to link to a ps:contact.	Deprecated: True
:url	*inet:url*	URL relevant for the software.	
:isos	*bool*	Set to True if the software is an operating system.	
:islib	*bool*	Set to True if the software is a library.	
:techniques	*array* type: *ou:technique* sorted: True uniq: True	Deprecated for scalability. Please use -(uses)> ou:technique.	Deprecated: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
it:prod:so	-(uses)>	ou:technique	The software uses the technique.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:soft:taxonomy

A software type taxonomy.

The base type for the form can be found at *it:prod:soft:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*it:prod:soft:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 284 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softfile

A file is distributed by a specific software version.

The base type for the form can be found at *it:prod:softfile*.

Properties:

name	type	doc	opts
:soft	*it:prod:softver*	The software which distributes the file.	Read Only: True
:file	*file:bytes*	The file distributed by the software.	Read Only: True
:path	*file:path*	The default installation path of the file.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 285 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softid

An identifier issued to a given host by a specific software application.

The base type for the form can be found at *it:prod:softid*.

Properties:

name	type	doc
:id	*str*	The ID issued by the software to the host.
:host	*it:host*	The host which was issued the ID by the software.
:soft	*it:prod:softver*	The software which issued the ID to the host.
:soft:name	*it:prod:softname*	The name of the software which issued the ID to the host.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softlib

A software version contains a library software version.

The base type for the form can be found at *it:prod:softlib*.

Properties:

name	type	doc	opts
:soft	*it:prod:softver*	The software version that contains the library.	Read Only: True
:lib	*it:prod:softver*	The library software version.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 287 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softname

A software product name.

The base type for the form can be found at *it:prod:softname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next page

Table 288 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softos

The software version is known to be compatible with the given os software version.

The base type for the form can be found at *it:prod:softos*.

Properties:

name	type	doc	opts
:soft	*it:prod:softver*	The software which can run on the operating system.	Read Only: True
:os	*it:prod:softver*	The operating system which the software can run on.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 289 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softreg

A registry entry is created by a specific software version.

The base type for the form can be found at *it:prod:softreg*.

Properties:

name	type	doc	opts
:softver	*it:prod:softver*	The software which creates the registry entry.	Read Only: True
:regval	*it:dev:regval*	The registry entry created by the software.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 290 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:prod:softver

A specific version of a software product.

The base type for the form can be found at *it:prod:softver*.

Properties:

name	type	doc	opts
:software	*it:prod:soft*	Software associated with this version instance.	
:software:name	*str* lower: True strip: True	Deprecated. Please use it:prod:softver:name.	Deprecated: True
:name	*it:prod:softname*	Name of the software version.	
:names	*array* type: *it:prod:softname* uniq: True sorted: True	Observed/variant names for this software version.	
:desc	*str*	A description of the software.	Display: {'hint': 'text'}
:cpe	*it:sec:cpe*	The NIST CPE 2.3 string specifying this software version.	
:cves	*array* type: *it:sec:cve* uniq: True sorted: True	A list of CVEs that apply to this software version.	
:vers	*it:dev:str*	Version string associated with this version instance.	
:vers:norm	*str* lower: True	Normalized version of the version string.	
:arch	*it:dev:str*	Software architecture.	
:released	*time*	Timestamp for when this version of the software was released.	
:semver	*it:semver*	System normalized semantic version number.	
:semver:major	*int*	Version major number.	
:semver:minor	*int*	Version minor number.	
:semver:patch	*int*	Version patch number.	
:semver:pre	*str*	Semver prerelease string.	
:semver:build	*str*	Semver build string.	
:url	*inet:url*	URL where a specific version of the software is available	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:query

A unique query string.

The base type for the form can be found at *it:query*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:reveng:filefunc

An instance of a function in an executable.

The base type for the form can be found at *it:reveng:filefunc*.

Properties:

name	type	doc	opts
:function	*it:reveng:function*	The guid matching the function.	Read Only: True
:file	*file:bytes*	The file that contains the function.	Read Only: True
:va	*int*	The virtual address of the first codeblock of the function.	
:rank	*int*	The function rank score used to evaluate if it exhibits interesting behavior.	
:complexity	*int*	The complexity of the function.	
:funccalls	*array* type: *it:reveng:filefunc* uniq: True sorted: True	Other function calls within the scope of the function.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 293 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:reveng:funcstr

A reference to a string inside a function.

The base type for the form can be found at *it:reveng:funcstr*.

Properties:

name	type	doc	opts
:function	*it:reveng:function*	The guid matching the function.	Read Only: True
:string	*str*	The string that the function references.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:reveng:function

A function inside an executable.

The base type for the form can be found at *it:reveng:function*.

Properties:

name	type	doc
:name	*str*	The name of the function.
:description	*str*	Notes concerning the function.
:impcalls	*array* type: *it:reveng:impfunc* uniq: True sorted: True	Calls to imported library functions within the scope of the function.
:strings	*array* type: *it:dev:str* uniq: True	An array of strings referenced within the function.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 295 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:reveng:impfunc

A function from an imported library.

The base type for the form can be found at *it:reveng:impfunc*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 296 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:screenshot

A screenshot of a host.

The base type for the form can be found at *it:screenshot*.

Properties:

name	type	doc	opts
:image	*file:bytes*	The image file.	
:desc	*str*	A brief description of the screenshot.	Display: {'hint': 'text'}
:exe	*file:bytes*	The executable file which caused the activity.	
:proc	*it:exec:proc*	The host process which caused the activity.	
:thread	*it:exec:thre*	The host thread which caused the activity.	
:host	*it:host*	The host on which the activity occurred.	
:time	*time*	The time that the activity started.	
:sandbox:f:	*file:bytes*	The initial sample given to a sandbox environment to analyze.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None

continues on next page

Table 297 – continued from previous page

source	verb	target	doc
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:c2:config

An extracted C2 config from an executable.

The base type for the form can be found at *it:sec:c2:config*.

Properties:

name	type	doc
:family	*it:prod:softname*	The name of the software family which uses the config.
:file	*file:bytes*	The file that the C2 config was extracted from.
:decoys	*array* type: *inet:url*	An array of URLs used as decoy connections to obfuscate the C2 servers.
:servers	*array* type: *inet:url*	An array of connection URLs built from host/port/passwd combinations.
:proxies	*array* type: *inet:url*	An array of proxy URLs used to communicate with the C2 server.
:listens	*array* type: *inet:url*	An array of listen URLs that the software should bind.
:dns:resolvers	*array* type: *inet:server*	An array of inet:servers to use when resolving DNS names.
:mutex	*it:dev:mutex*	The mutex that the software uses to prevent multiple-installations.
:campaigncode	*it:dev:str*	The operator selected string used to identify the campaign or group of targets.
:crypto:key	*crypto:key*	Static key material used to encrypt C2 communications.
:connect:delay	*duration*	The time delay from first execution to connecting to the C2 server.
:connect:interval	*duration*	The configured duration to sleep between connections to the C2 server.
:raw	*data*	A JSON blob containing the raw config extracted from the binary.
:http:headers	*array* type: *inet:http:header*	An array of HTTP headers that the sample should transmit to the C2 server.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:cpe

A NIST CPE 2.3 Formatted String.

The base type for the form can be found at *it:sec:cpe*.

Properties:

name	type	doc	opts
:v2_2	*it:sec:cpe:v2_2*	The CPE 2.2 string which is equivalent to the primary property.	
:part	*str* lower: True strip: True	The "part" field from the CPE 2.3 string.	Read Only: True
:vendor	*ou:name*	The "vendor" field from the CPE 2.3 string.	Read Only: True
:product	*str* lower: True strip: True	The "product" field from the CPE 2.3 string.	Read Only: True
:version	*str* lower: True strip: True	The "version" field from the CPE 2.3 string.	Read Only: True
:update	*str* lower: True strip: True	The "update" field from the CPE 2.3 string.	Read Only: True
:edition	*str* lower: True strip: True	The "edition" field from the CPE 2.3 string.	Read Only: True
:language	*str* lower: True strip: True	The "language" field from the CPE 2.3 string.	Read Only: True
:sw_edition	*str* lower: True strip: True	The "sw_edition" field from the CPE 2.3 string.	Read Only: True
:target_sw	*str* lower: True strip: True	The "target_sw" field from the CPE 2.3 string.	Read Only: True
:target_hw	*str* lower: True strip: True	The "target_hw" field from the CPE 2.3 string.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:cve

A vulnerability as designated by a Common Vulnerabilities and Exposures (CVE) number.

The base type for the form can be found at *it:sec:cve*.

An example of `it:sec:cve`:

- `cve-2012-0158`

Properties:

name	type	doc	opts
:desc	*str*	Deprecated. Please use risk:vuln:cve:desc.	Deprecated: True
:url	*inet:url*	Deprecated. Please use risk:vuln:cve:url.	Deprecated: True
:references	*array* type: *inet:url* uniq: True sorted: True	Deprecated. Please use risk:vuln:cve:referen	Deprecated: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next

Table 300 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:cwe

NIST NVD Common Weaknesses Enumeration Specification.

The base type for the form can be found at *it:sec:cwe*.

An example of `it:sec:cwe`:

- `CWE-120`

  Properties:

name	type	doc	opts
:name	*str*	The CWE description field.	Example: `Buffer Copy without Checking Size of Input (Classic Buffer Overflow)`
:desc	*str*	The CWE description field.	Display: `{'hint': 'text'}`
:url	*inet:url*	A URL linking this CWE to a full description.	
:parents	*array* type: *it:sec:cwe* uniq: `True` sorted: `True` split: `,`	An array of ChildOf CWE Relationships.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next

Table 301 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:metrics

A node used to track metrics of an organization's infosec program.

The base type for the form can be found at *it:sec:metrics*.

Properties:

name	type	doc
:org	*ou:org*	The organization whose security program is being measured.
:org:name	*ou:nan*	The organization name. Used for entity resolution.
:org:fqdn	*inet:fq*	The organization FQDN. Used for entity resolution.
:period	*ival*	The time period used to compute the metrics.
:alerts:meantime:tri	*duration*	The mean time to triage alerts generated within the time period.
:alerts:count	*int*	The total number of alerts generated within the time period.
:alerts:falsepos	*int*	The number of alerts generated within the time period that were determined to be false positives.
:assets:hosts	*int*	The total number of hosts within scope for the information security program.
:assets:users	*int*	The total number of users within scope for the information security program.
:assets:vulns:count	*int*	The number of asset vulnerabilities being tracked at the end of the time period.
:assets:vulns:preexi	*int*	The number of asset vulnerabilities being tracked at the beginning of the time period.
:assets:vulns:discov	*int*	The number of asset vulnerabilities discovered during the time period.
:assets:vulns:mitiga	*int*	The number of asset vulnerabilities mitigated during the time period.
:assets:vulns:meanti	*duration*	The mean time to mitigate for vulnerable assets mitigated during the time period.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:stix:bundle

A STIX bundle.

The base type for the form can be found at *it:sec:stix:bundle*.

Properties:

name	type	doc
:id	*str*	The id field from the STIX bundle.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.

continues on next page

Table 303 – continued from previous page

source	verb	target	doc
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:stix:indicator

A STIX indicator pattern.

The base type for the form can be found at *it:sec:stix:indicator*.

Properties:

name	type	doc
:id	*str*	The STIX id field from the indicator pattern.
:name	*str*	The name of the STIX indicator pattern.
:pattern	*str*	The STIX indicator pattern text.
:created	*time*	The time that the indicator pattern was first created.
:updated	*time*	The time that the indicator pattern was last modified.
:labels	*array* type: *str* uniq: True sorted: True	The label strings embedded in the STIX indicator pattern.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.

continues on next page

Table 304 – continued from previous page

source	verb	target	doc
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:vuln:scan

An instance of running a vulnerability scan.

The base type for the form can be found at *it:sec:vuln:scan*.

Properties:

name	type	doc	opts
:time	*time*	The time that the scan was started.	
:desc	*str*	Description of the scan and scope.	Display: {'hint': 'text'}
:ext:id	*str*	An externally generated ID for the scan.	
:ext:url	*inet:url*	An external URL which documents the scan.	
:software	*it:prod:softver*	The scanning software used.	
:software:na	*it:prod:softnar*	The name of the scanner software.	
:operator	*ps:contact*	Contact information for the scan operator.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

it:sec:vuln:scan:result

A vulnerability scan result for an asset.

The base type for the form can be found at *it:sec:vuln:scan:result*.

Properties:

name	type	doc
:scan	*it:sec:vuln:sc*	The scan that discovered the vulnerability in the asset.
:vuln	*risk:vuln*	The vulnerability detected in the asset.
:asset	*ndef*	The node which is vulnerable.
:desc	*str*	A description of the vulnerability and how it was detected in the asset.
:time	*time*	The time that the scan result was produced.
:ext:id	*str*	An externally generated ID for the scan result.
:ext:url	*inet:url*	An external URL which documents the scan result.
:mitigatic	*risk:mitigatio*	The mitigation used to address this asset vulnerability.
:mitigatec	*time*	The time that the vulnerability in the asset was mitigated.
:priority	*meta:priority*	The priority of mitigating the vulnerability.
:severity	*meta:severity*	The severity of the vulnerability in the asset. Use "none" for no vulnerability discovered.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 306 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

lang:idiom

Deprecated. Please use lang:translation.

The base type for the form can be found at *lang:idiom*.

Properties:

name	type	doc	opts
:url	*inet:url*	Authoritative URL for the idiom.	
:desc:en	*str*	English description.	Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 307 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

lang:language

A specific written or spoken language.

The base type for the form can be found at *lang:language*.

Properties:

name	type	doc
:code	*lang:code*	The language code for this language.
:name	*lang:name*	The primary name of the language.
:names	*array* type: *lang:name* sorted: True uniq: True	An array of alternative names for the language.
:skill	*ps:skill*	The skill used to annotate proficiency in the language.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

lang:name

A name used to refer to a language.

The base type for the form can be found at *lang:name*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

lang:trans

Deprecated. Please use lang:translation.

The base type for the form can be found at *lang:trans*.

Properties:

name	type	doc	opts
`:text:en`	*str*	English translation.	Display: `{'hint': 'text'}`
`:desc:en`	*str*	English description.	Display: `{'hint': 'text'}`

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requiremen`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)>`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target node.
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	`*`	The meta:rule is designed to detect instances of the target node.
`meta:rule`	`-(matches)>`	`*`	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	`*`	The meta:source observed the target node.
`ou:campaign`	`-(targets)>`	`*`	The campaign targeted the target nodes.
`ou:campaign`	`-(uses)>`	`*`	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	`*`	The contribution includes the specific node.
`ou:org`	`-(has)>`	`*`	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	`*`	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	`*`	The organization targets the target node.
`ou:org`	`-(uses)>`	`*`	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	`*`	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	`*`	The contact owns or owned the target node.
`ps:person`	`-(has)>`	`*`	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	`*`	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	`*`	The attack targeted the target node.
`risk:attack`	`-(uses)>`	`*`	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	`*`	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	`*`	The extortion event was based on attacker access to the target node

continues on next page

Table 310 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

lang:translation

A translation of text from one language to another.

The base type for the form can be found at *lang:translation*.

Properties:

name	type	doc	opts
:input	*str*	The input text.	Example: hola
:input:lang	*lang:code*	The input language code.	
:output	*str*	The output text.	Example: hi
:output:lan	*lang:code*	The output language code.	
:desc	*str*	A description of the meaning of the output.	Example: A standard greeting
:engine	*it:prod:softve*	The translation engine version used.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 311 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

mat:item

A GUID assigned to a material object.

The base type for the form can be found at *mat:item*.

Properties:

name	type	doc
:name	*str* lower: True	The name of the material item.
:type	*mat:type*	The taxonomy type of the item.
:spec	*mat:spec*	The specification which defines this item.
:place	*geo:place*	The most recent place the item is known to reside.
:latlong	*geo:latlong*	The last known lat/long location of the node.
:loc	*loc*	The geo-political location string for the node.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

mat:itemimage

The base type for compound node fields.

The base type for the form can be found at *mat:itemimage*.

Properties:

name	type	doc	opts
:item	*mat:item*	The item contained within the image file.	Read Only: True
:file	*file:bytes*	The file containing an image of the item.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.

continues on next page

Table 313 – continued from previous page

source	verb	target	doc
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

mat:spec

A GUID assigned to a material specification.

The base type for the form can be found at *mat:spec*.

Properties:

name	type	doc
:name	*str* lower: True	The name of the material specification.
:type	*mat:type*	The taxonomy type for the specification.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 314 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

mat:specimage

The base type for compound node fields.

The base type for the form can be found at *mat:specimage*.

Properties:

name	type	doc	opts
:spec	*mat:spec*	The spec contained within the image file.	Read Only: True
:file	*file:bytes*	The file containing an image of the spec.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.

continues on next page

Table 315 – continued from previous page

source	verb	target	doc
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

media:news

A GUID for a news article or report.

The base type for the form can be found at *media:news*.

Properties:

name	type	doc	opts
`:url`	*inet:url*	The (optional) URL where the news was published.	Example: `http://cnn.com/news/mars-lander.html`
`:url:fqdn`	*inet:fqdn*	The FQDN within the news URL.	Example: `cnn.com`
`:type`	*media:news:taxonomy*	A taxonomy for the type of reporting or news.	
`:file`	*file:bytes*	The (optional) file blob containing or published as the news.	
`:title`	*str* lower: `True`	Title/Headline for the news.	Example: `mars lander reaches mars` Display: `{'hint': 'text'}`
`:summary`	*str*	A brief summary of the news item.	Example: `lorum ipsum` Display: `{'hint': 'text'}`
`:publisher`	*ou:org*	The organization which published the news.	
`:publisher:name`	*ou:name*	The name of the publishing org used to publish the news.	
`:published`	*time*	The date the news item was published.	Example: `20161201180433`
`:updated`	*time* ismax: `True`	The last time the news item was updated.	Example: `20161201180433`
`:org`	*ou:alias*	Deprecated. Please use :publisher:name.	Deprecated: `True`
`:author`	*ps:name*	Deprecated. Please use :authors array of ps:contact nodes.	Deprecated: `True`
`:authors`	*array* type: *ps:contact* split: `,` uniq: `True` sorted: `True`	An array of authors of the news item.	
`:rss:feed`	*inet:url*	The RSS feed that published the news.	
`:ext:id`	*str*	An external identifier specified by the publisher.	
`:topics`		An array of relevant	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

media:news:taxonomy

A taxonomy of types or sources of news.

The base type for the form can be found at *media:news:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*media:news:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 317 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

media:topic

A topic string.

The base type for the form can be found at *media:topic*.

Properties:

name	type	doc
:desc	*str*	A brief description of the topic.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 318 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:event

An analytically relevant event in a curated timeline.

The base type for the form can be found at *meta:event*.

Properties:

name	type	doc	opts
:timeline	*meta:timeline*	The timeline containing the event.	
:title	*str*	A title for the event.	
:summary	*str*	A prose summary of the event.	Display: {'hint': 'text'}
:time	*time*	The time that the event occurred.	
:duration	*duration*	The duration of the event.	
:type	*meta:event:taxonom*	Type of event.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:event:taxonomy

A taxonomy of event types for meta:event nodes.

The base type for the form can be found at *meta:event:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*meta:event:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 320 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:note

An analyst note about nodes linked with -(about)> edges.

The base type for the form can be found at *meta:note*.

Properties:

name	type	doc	opts
:type	*meta:note:type:ta*	The note type.	
:text	*str*	The analyst authored note text.	Display: {'hint': 'text', 'syntax': 'markdown'}
:autho:	*ps:contact*	The contact information of the author.	
:creat	*syn:user*	The synapse user who authored the note.	
:creat	*time*	The time the note was created.	
:updat	*time*	The time the note was updated.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremer	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat):	geo:telem	The source node was seen at the geo:telem node place and time.
meta:note	-(about)>	*	The meta:note is about the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:note:type:taxonomy

An analyst note type taxonomy.

The base type for the form can be found at *meta:note:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*meta:note:type:taxonon*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 322 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:rule

A generic rule linked to matches with -(matches)> edges.

The base type for the form can be found at *meta:rule*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A name for the rule.	
:desc	*str*	A description of the rule.	Display: {'hint': 'text'}
:text	*str*	The text of the rule logic.	Display: {'hint': 'text'}
:author	*ps:contact*	The contact information of the rule author.	
:created	*time*	The time the rule was initially created.	
:updated	*time*	The time the rule was most recently modified.	
:url	*inet:url*	A URL which documents the rule.	
:ext:id	*str*	An external identifier for the rule.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremer	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
meta:rule	-(detects)	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)	*	The meta:rule has matched on target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target noc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query
meta:note	-(about)>	*	The meta:note is about the target node.
meta:ruleset	-(has)>	meta:rule	The meta:ruleset includes the meta:rule.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:ruleset

A set of rules linked with -(has)> edges.

The base type for the form can be found at *meta:ruleset*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A name for the ruleset.	
:desc	*str*	A description of the ruleset.	Display: {'hint': 'text'}
:author	*ps:contact*	The contact information of the ruleset author.	
:created	*time*	The time the ruleset was initially created.	
:updated	*time*	The time the ruleset was most recently modified.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
meta:rulese	-(has)>	meta:rule	The meta:ruleset includes the meta:rule.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 324 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:seen

Annotates that the data in a node was obtained from or observed by a given source.

The base type for the form can be found at *meta:seen*.

Properties:

name	type	doc	opts
:source	*meta:source*	The source which observed or provided the node.	Read Only: True
:node	*ndef*	The node which was observed by or received from the source.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:source

A data source unique identifier.

The base type for the form can be found at *meta:source*.

Properties:

name	type	doc
:name	*str* lower: True	A human friendly name for the source.
:type	*str* lower: True	An optional type field used to group sources.
:url	*inet:url*	A URL which documents the meta source.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
meta:sourc	-(seen)>	*	The meta:source observed the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 326 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:timeline

A curated timeline of analytically relevant events.

The base type for the form can be found at *meta:timeline*.

Properties:

name	type	doc	opts
:title	*str*	A title for the timeline.	Example: The history of the Vertex Project
:summar	*str*	A prose summary of the timeline.	Display: {'hint': 'text'}
:type	*meta:timeline:taxo*	The type of timeline.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 327 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

meta:timeline:taxonomy

A taxonomy of timeline types for meta:timeline nodes.

The base type for the form can be found at *meta:timeline:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: `True` Display: `{'hint': 'text'}`
:desc	*str*	A definition of the taxonomy entry.	Display: `{'hint': 'text'}`
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: `True`
:depth	*int*	The depth indexed from 0.	Read Only: `True`
:parent	*meta:timeline:taxonom*	The taxonomy parent.	Read Only: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:attendee

A node representing a person attending a meeting, conference, or event.

The base type for the form can be found at *ou:attendee*.

Properties:

name	type	doc
:person	*ps:contact*	The contact information for the person who attended the event.
:arrived	*time*	The time when the person arrived.
:departed	*time*	The time when the person departed.
:roles	*array* type: *ou:role* split: , uniq: True sorted: True	List of the roles the person had at the event.
:meet	*ou:meet*	The meeting that the person attended.
:conference	*ou:conference*	The conference that the person attended.
:conference:event	*ou:conference:event*	The conference event that the person attended.
:contest	*ou:contest*	The contest that the person attended.
:preso	*ou:preso*	The presentation that the person attended.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 329 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:award

An award issued by an organization.

The base type for the form can be found at *ou:award*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name of the award.	Example: Bachelors of Science
:type	*str* lower: True onespace: True	The type of award.	Example: certification
:org	*ou:org*	The organization which issues the award.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 330 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:campaign

Represents an org's activity in pursuit of a goal.

The base type for the form can be found at *ou:campaign*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org carrying out the campaign.	
:org:name	*ou:name*	The name of the org responsible for the campaign. Used for entity resolution.	
:org:fqdn	*inet:fqdn*	The FQDN of the org responsible for the campaign. Used for entity resolution.	
:goal	*ou:goal*	The assessed primary goal of the campaign.	
:actors	*array* type: *ps:contact* split: `,` uniq: `True` sorted: `True`	Actors who participated in the campaign.	
:goals	*array* type: *ou:goal* split: `,` uniq: `True` sorted: `True`	Additional assessed goals of the campaign.	
:success	*bool*	Records the success/failure status of the campaign if known.	
:name	*ou:campname*	A terse name of the campaign.	Example: `operation overlord`
:names	*array* type: *ou:campname* sorted: `True` uniq: `True`	An array of alternate names for the campaign.	
:reporter	*ou:org*	The organization reporting on the campaign.	
:reporter:name	*ou:name*	The name of the organization reporting on the campaign.	
:type	*str*	Deprecated. Use the :camptype taxonomy.	Deprecated: `True`
:sophistication	*meta:sophistication*	The assessed sophistication of the campaign.	
:timeline	*meta:timeline*	A timeline of significant events related to the campaign.	
:camptype	*ou:camptype*	The campaign type taxonomy.	Display: `{'hint': 'taxonomy'}`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme:	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
ou:campaig	-(targets)	*	The campaign targeted the target nodes.
ou:campaig	-(uses)>	ou:technique	The campaign used the technique.
ou:campaig	-(uses)>	*	The campaign made use of the target node.

Target Edges:

source	verb	tar-get	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:campname

A campaign name.

The base type for the form can be found at *ou:campname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:camptype

An campaign type taxonomy.

The base type for the form can be found at *ou:camptype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:camptype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 332 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conference

A conference with a name and sponsoring org.

The base type for the form can be found at *ou:conference*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org which created/managed the conference.	
:organizer	*ps:contact*	Contact information for the primary organizer of the conference.	
:sponsors	*array* type: *ps:contact* uniq: True sorted: True	An array of contacts which sponsored the conference.	
:name	*str* lower: True	The full name of the conference.	Example: decfon 2017
:desc	*str* lower: True	A description of the conference.	Example: annual cybersecurity conference Display: {'hint': 'text'}
:base	*str* lower: True strip: True	The base name which is shared by all conference instances.	Example: defcon
:start	*time*	The conference start date / time.	
:end	*time*	The conference end date / time.	
:place	*geo:place*	The geo:place node where the conference was held.	
:url	*inet:url*	The inet:url node for the conference website.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conference:attendee

Deprecated. Please use ou:attendee.

The base type for the form can be found at *ou:conference:attendee*.

Properties:

name	type	doc	opts
:conference	*ou:conference*	The conference which was attended.	Read Only: True
:person	*ps:person*	The person who attended the conference.	Read Only: True
:arrived	*time*	The time when a person arrived to the conference.	
:departed	*time*	The time when a person departed from the conference.	
:role:staff	*bool*	The person worked as staff at the conference.	
:role:speaker	*bool*	The person was a speaker or presenter at the conference.	
:roles	*array* type: *str* uniq: True sorted: True	List of the roles the person had at the conference.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 334 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conference:event

A conference event with a name and associated conference.

The base type for the form can be found at *ou:conference:event*.

Properties:

name	type	doc	opts
:conference	*ou:conference*	The conference to which the event is associated.	Read Only: True
:organizer	*ps:contact*	Contact information for the primary organizer of the event.	
:sponsors	*array* type: *ps:contact* uniq: True sorted: True	An array of contacts which sponsored the event.	
:place	*geo:place*	The geo:place where the event occurred.	
:name	*str* lower: True	The name of the conference event.	Example: foobar conference dinner
:desc	*str* lower: True	A description of the conference event.	Example: foobar conference networking dinner at ridge hotel Display: {'hint': 'text'}
:url	*inet:url*	The inet:url node for the conference event website.	
:contact	*ps:contact*	Contact info for the event.	
:start	*time*	The event start date / time.	
:end	*time*	The event end date / time.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conference:event:attendee

Deprecated. Please use ou:attendee.

The base type for the form can be found at *ou:conference:event:attendee*.

Properties:

name	type	doc	opts
:event	*ou:conference:event*	The conference event which was attended.	Read Only: True
:person	*ps:person*	The person who attended the conference event.	Read Only: True
:arrived	*time*	The time when a person arrived to the conference event.	
:departed	*time*	The time when a person departed from the conference event.	
:roles	*array* type: *str* uniq: True sorted: True	List of the roles the person had at the conference event.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 336 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conflict

Represents a conflict where two or more campaigns have mutually exclusive goals.

The base type for the form can be found at *ou:conflict*.

Properties:

name	type	doc
:name	*str* onespace: True	The name of the conflict.
:started	*time*	The time the conflict began.
:ended	*time*	The time the conflict ended.
:timeline	*meta:timeline*	A timeline of significant events related to the conflict.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.

continues on next page

Table 337 – continued from previous page

source	verb	target	doc
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:contest

A competitive event resulting in a ranked set of participants.

The base type for the form can be found at *ou:contest*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name of the contest.	Example: defcon ctf 2020
:type	*str* lower: True onespace: True	The type of contest.	Example: cyber ctf
:family	*str* lower: True onespace: True	A name for a series of recurring contests.	Example: defcon ctf
:desc	*str* lower: True	A description of the contest.	Example: the capture-the-flag event hosted at defcon 2020 Display: {'hint': 'text'}
:url	*inet:url*	The contest website URL.	
:start	*time*	The contest start date / time.	
:end	*time*	The contest end date / time.	
:loc	*loc*	The geopolitical affiliation of the contest.	
:place	*geo:place*	The geo:place where the contest was held.	
:latlong	*geo:latlong*	The latlong where the contest was held.	
:conference	*ou:conference*	The conference that the contest is associated with.	
:contests	*array* type: *ou:contest* split: , uniq: True sorted: True	An array of sub-contests that contributed to the rankings.	
:sponsors	*array* type: *ps:contact* split: , uniq: True sorted: True	Contact information for contest sponsors.	
:organizers		Contact information	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:contest:result

The results from a single contest participant.

The base type for the form can be found at *ou:contest:result*.

Properties:

name	type	doc	opts
:contest	*ou:contest*	The contest.	Read Only: True
:participant	*ps:contact*	The participant.	Read Only: True
:rank	*int*	The rank order of the participant.	
:score	*int*	The score of the participant.	
:url	*inet:url*	The contest result website URL.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 339 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:contract

An contract between multiple entities.

The base type for the form can be found at *ou:contract*.

Properties:

name	type	doc	opts
:title	*str*	A terse title for the contract.	
:type	*ou:conttype*	The type of contract.	
:sponsor	*ps:contact*	The contract sponsor.	
:parties	*array* type: *ps:contact* uniq: `True` sorted: `True`	The non-sponsor entities bound by the contract.	
:document	*file:bytes*	The best/current contract document.	
:signed	*time*	The date that the contract signing was complete.	
:begins	*time*	The date that the contract goes into effect.	
:expires	*time*	The date that the contract expires.	
:completed	*time*	The date that the contract was completed.	
:terminated	*time*	The date that the contract was terminated.	
:award:price	*econ:price*	The value of the contract at time of award.	
:budget:price	*econ:price*	The amount of money budgeted for the contract.	
:currency	*econ:currency*	The currency of the econ:price values.	
:purchase	*econ:purchase*	Purchase details of the contract.	
:requirements	*array* type: *ou:goal* uniq: `True` sorted: `True`	The requirements levied upon the parties.	
:types	*array* type: *ou:contract:type* split: `,` uniq: `True` sorted: `True`	A list of types that apply to the contract.	Deprecated: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:contribution

Represents a specific instance of contributing material support to a campaign.

The base type for the form can be found at *ou:contribution*.

Properties:

name	type	doc
:from	*ps:contact*	The contact information of the contributor.
:campaign	*ou:campaign*	The campaign receiving the contribution.
:value	*econ:price*	The assessed value of the contribution.
:currency	*econ:currency*	The currency used for the assessed value.
:time	*time*	The time the contribution occurred.
:material:spec	*mat:spec*	The specification of material items contributed.
:material:count	*int*	The number of material items contributed.
:monetary:payment	*econ:acct:payment*	Payment details for a monetary contribution.
:personnel:count	*int*	Number of personnel contributed to the campaign.
:personnel:jobtitle	*ou:jobtitle*	Title or designation for the contributed personnel.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
ou:contribut:	-(includes	*	The contribution includes the specific node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 341 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:conttype

A contract type taxonomy.

The base type for the form can be found at *ou:conttype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:conttype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:employment

An employment type taxonomy.

The base type for the form can be found at *ou:employment*.

An example of `ou:employment`:

- `fulltime.salary`

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:employment*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next

Table 343 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:goal

An assessed or stated goal which may be abstract or org specific.

The base type for the form can be found at *ou:goal*.

Properties:

name	type	doc	opts
:name	*ou:goalname*	A terse name for the goal.	
:names	*array* type: *ou:goalname* sorted: True uniq: True	An array of alternate names for the goal. Used to merge/resolve goals.	
:type	*ou:goal:type:taxonomy*	A type taxonomy entry for the goal.	
:desc	*str*	A description of the goal.	Display: {'hint': 'text'}
:prev	*ou:goal*	Deprecated. Please use ou:goal:type taxonomy.	Deprecated: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:goal:type:taxonomy

A taxonomy of goal types.

The base type for the form can be found at *ou:goal:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:goal:type:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 345 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:goalname

A goal name.

The base type for the form can be found at *ou:goalname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 346 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:hasalias

The knowledge that an organization has an alias.

The base type for the form can be found at *ou:hasalias*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org guid which has the alias.	Read Only: True
:alias	*ou:alias*	Alias for the organization.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 347 – continued from previous page

source	verb	target	doc
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:hasgoal

Deprecated. Please use ou:org:goals.

The base type for the form can be found at *ou:hasgoal*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org which has the goal.	Read Only: True
:goal	*ou:goal*	The goal which the org has.	Read Only: True
:stated	*bool*	Set to true/false if the goal is known to be self stated.	
:window	*ival*	Set if a goal has a limited time window.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:id:number

A unique id number issued by a specific organization.

The base type for the form can be found at *ou:id:number*.

Properties:

name	type	doc	opts
:type	*ou:id:type*	The type of org id.	Read Only: True
:value	*ou:id:value*	The value of org id.	Read Only: True
:status	*str* lower: True strip: True	A freeform status such as valid, suspended, expired.	
:issued	*time*	The time at which the org issued the ID number.	
:expires	*time*	The time at which the ID number expires.	
:issuer	*ps:contact*	The contact information of the office which issued the ID number.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 349 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:id:type

A type of id number issued by an org.

The base type for the form can be found at *ou:id:type*.

Properties:

name	type	doc
:org	*ou:org*	The org which issues id numbers of this type.
:name	*str*	The friendly name of the id number type.
:url	*inet:url*	The official URL of the issuer.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 350 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:id:update

A status update to an org:id:number.

The base type for the form can be found at *ou:id:update*.

Properties:

name	type	doc
:number	*ou:id:number*	The id number that was updated.
:status	*str* strip: True lower: True	The updated status of the id number.
:time	*time*	The date/time that the id number was updated.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:industry

An industry classification type.

The base type for the form can be found at *ou:industry*.

Properties:

name	type	doc	opts
:name	*ou:industryname*	The name of the industry.	
:type	*ou:industry:type:taxon*	A taxonomy entry for the industry.	
:names	*array* type: *ou:industryname* uniq: True sorted: True	An array of alternative names for the industry.	
:subs	*array* type: *ou:industry* split: , uniq: True sorted: True	Deprecated. Please use ou:industry:type taxonomy.	Deprecated: True
:sic	*array* type: *ou:sic* split: , uniq: True sorted: True	An array of SIC codes that map to the industry.	
:naics	*array* type: *ou:naics* split: , uniq: True sorted: True	An array of NAICS codes that map to the industry.	
:isic	*array* type: *ou:isic* split: , uniq: True sorted: True	An array of ISIC codes that map to the industry.	
:desc	*str*	A description of the industry.	Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target no
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target n
it:exec:query	-(found)>	*	The target node was returned as a result of running the que
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	ou:industry	The attack targeted the industry.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the com
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the tar
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	ou:industry	The threat cluster targets the industry.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:industryname

The name of an industry.

The base type for the form can be found at *ou:industryname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:jobtitle

A title for a position within an org.

The base type for the form can be found at *ou:jobtitle*.

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requiremen`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)>`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target node.
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	`*`	The meta:rule is designed to detect instances of the target node.
`meta:rule`	`-(matches)>`	`*`	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	`*`	The meta:source observed the target node.
`ou:campaign`	`-(targets)>`	`*`	The campaign targeted the target nodes.
`ou:campaign`	`-(uses)>`	`*`	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	`*`	The contribution includes the specific node.
`ou:org`	`-(has)>`	`*`	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	`*`	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	`*`	The organization targets the target node.
`ou:org`	`-(uses)>`	`*`	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	`*`	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	`*`	The contact owns or owned the target node.
`ps:person`	`-(has)>`	`*`	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	`*`	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	`*`	The attack targeted the target node.
`risk:attack`	`-(uses)>`	`*`	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	`*`	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	`*`	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	`*`	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	`*`	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	`*`	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	`*`	The tool uses the target node.
`sci:evidence`	`-(has)>`	`*`	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	`*`	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	`*`	The observations are summarized from the target nodes.

ou:jobtype

A taxonomy of job types.

The base type for the form can be found at *ou:jobtype*.

An example of `ou:jobtype`:

- `it.dev.python`

Properties:

name	type	doc	opts
`:title`	*str*	A brief title of the definition.	
`:summary`	*str*	Deprecated. Please use title/desc.	Deprecated: `True` Display: `{'hint': 'text'}`
`:desc`	*str*	A definition of the taxonomy entry.	Display: `{'hint': 'text'}`
`:sort`	*int*	A display sort order for siblings.	
`:base`	*taxon*	The base taxon.	Read Only: `True`
`:depth`	*int*	The depth indexed from 0.	Read Only: `True`
`:parent`	*ou:jobtype*	The taxonomy parent.	Read Only: `True`

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.

continues on next

Table 355 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:meet

An informal meeting of people which has no title or sponsor. See also: ou:conference.

The base type for the form can be found at *ou:meet*.

Properties:

name	type	doc
:name	*str* lower: True	A human friendly name for the meeting.
:start	*time*	The date / time the meet starts.
:end	*time*	The date / time the meet ends.
:place	*geo:place*	The geo:place node where the meet was held.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:meet:attendee

Deprecated. Please use ou:attendee.

The base type for the form can be found at *ou:meet:attendee*.

Properties:

name	type	doc	opts
:meet	*ou:meet*	The meeting which was attended.	Read Only: True
:person	*ps:person*	The person who attended the meeting.	Read Only: True
:arrived	*time*	The time when a person arrived to the meeting.	
:departed	*time*	The time when a person departed from the meeting.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 357 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:member

Deprecated. Please use ou:position.

The base type for the form can be found at *ou:member*.

Properties:

name	type	doc	opts
:org	*ou:org*	The GUID of the org the person is a member of.	Read Only: True
:person	*ps:person*	The GUID of the person that is a member of an org.	Read Only: True
:title	*str* lower: True strip: True	The persons normalized title.	
:start	*time* ismin: True	Earliest known association of the person with the org.	
:end	*time* ismax: True	Most recent known association of the person with the org.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:name

The name of an organization. This may be a formal name or informal name of the organization.

The base type for the form can be found at *ou:name*.

An example of `ou:name`:

- `acme corporation`

  Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:opening

A job/work opening within an org.

The base type for the form can be found at *ou:opening*.

Properties:

name	type	doc
:org	*ou:org*	The org which has the opening.
:orgname	*ou:name*	The name of the organization as listed in the opening.
:orgfqdn	*inet:fqdn*	The FQDN of the organization as listed in the opening.
:posted	*time*	The date/time that the job opening was posted.
:removed	*time*	The date/time that the job opening was removed.
:postings	*array* type: *inet:url* uniq: `True` sorted: `True`	URLs where the opening is listed.
:contact	*ps:contact*	The contact details to inquire about the opening.
:loc	*loc*	The geopolitical boundary of the opening.
:jobtype	*ou:jobtype*	The job type taxonomy.
:employment	*ou:employment*	The type of employment.
:jobtitle	*ou:jobtitle*	The title of the opening.
:remote	*bool*	Set to true if the opening will allow a fully remote worker.
:yearlypay	*econ:price*	The yearly income associated with the opening.
:paycurrency	*econ:currency*	The currency that the yearly pay was delivered in.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:org

A GUID for a human organization such as a company or military unit.

The base type for the form can be found at *ou:org*.

Properties:

name	type	doc	opts
:loc	*loc*	Location for an organization.	
:name	*ou:name*	The localized name of an organization.	
:type	*str* lower: `True` strip: `True`	The type of organization.	Deprecated: `True`
:orgtype	*ou:orgtype*	The type of organization.	Display: `{'hint': 'taxonomy'}`
:vitals	*ou:vitals*	The most recent/accurate ou:vitals for the org.	
:desc	*str*	A description of the org.	Display: `{'hint': 'text'}`
:logo	*file:bytes*	An image file representing the logo for the organization.	
:names	*array* type: *ou:name* uniq: `True` sorted: `True`	A list of alternate names for the organization.	
:alias	*ou:alias*	The default alias for an organization.	
:phone	*tel:phone*	The primary phone number for the organization.	
:sic	*ou:sic*	The Standard Industrial Classification code for the organization.	Deprecated: `True`
:naics	*ou:naics*	The North American Industry Classification System code for the organization.	Deprecated: `True`
:industries	*array* type: *ou:industry* uniq: `True` sorted: `True`	The industries associated with the org.	
:us:cage	*gov:us:cage*	The Commercial and Government Entity (CAGE) code for the organization.	
:founded	*time*	The date on which the org was founded.	
:dissolved	*time*	The date on which the org was dissolved.	
:url	*inet:url*	The primary url for the organization.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	ou:technique	The org uses the technique.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:org:has

An org owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time.

The base type for the form can be found at *ou:org:has*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org who owns or controls the object or resource.	Read Only: True
:node	*ndef*	The object or resource that is owned or controlled by the org.	Read Only: True
:node:for	*str*	The form of the object or resource that is owned or controlled by the org.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

continues on next page

Table 361 – continued from previous page

source	verb	target	doc
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:orgnet4

An organization's IPv4 netblock.

The base type for the form can be found at *ou:orgnet4*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org guid which owns the netblock.	Read Only: True
:net	*inet:net4*	Netblock owned by the organization.	Read Only: True
:name	*str* lower: True strip: True	The name that the organization assigns to this netblock.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 362 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:orgnet6

An organization's IPv6 netblock.

The base type for the form can be found at *ou:orgnet6*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org guid which owns the netblock.	Read Only: True
:net	*inet:net6*	Netblock owned by the organization.	Read Only: True
:name	*str* lower: True strip: True	The name that the organization assigns to this netblock.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:orgtype

An org type taxonomy.

The base type for the form can be found at *ou:orgtype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:orgtype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 364 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:position

A position within an org. May be organized into an org chart.

The base type for the form can be found at *ou:position*.

Properties:

name	type	doc
:org	*ou:org*	The org which has the position.
:team	*ou:team*	The team that the position is a member of.
:contact	*ps:contact*	The contact info for the person who holds the position.
:title	*str* lower: True onespace: True	The title of the position.
:reports	*array* type: *ou:position* uniq: True sorted: True	An array of positions which report to this position.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:preso

A webinar, conference talk, or other type of presentation.

The base type for the form can be found at *ou:preso*.

Properties:

name	type	doc	opts
:organizer	*ps:contact*	Contact information for the primary organizer of the presentation.	
:sponsors	*array* type: *ps:contact* uniq: `True` sorted: `True`	A set of contacts which sponsored the presentation.	
:presenters	*array* type: *ps:contact* uniq: `True` sorted: `True`	A set of contacts which gave the presentation.	
:title	*str* lower: `True`	The full name of the presentation.	Example: `Synapse 101 - 2021/06/22`
:desc	*str* lower: `True`	A description of the presentation.	Display: `{'hint': 'text'}`
:time	*time*	The scheduled presentation start time.	
:duration	*duration*	The scheduled duration of the presentation.	
:loc	*loc*	The geopolitical location string for where the presentation was given.	
:place	*geo:place*	The geo:place node where the presentation was held.	
:deck:url	*inet:url*	The URL hosting a copy of the presentation materials.	
:deck:file	*file:bytes*	A file containing the presentation materials.	
:attendee:url	*inet:url*	The URL visited by live attendees of the presentation.	
:recording:url	*inet:url*	The URL hosting a recording of the presentation.	
:recording:file	*file:bytes*	A file containing a recording of the presentation.	
:conference	*ou:conference*	The conference which hosted the presentation.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:requirement

A specific requirement.

The base type for the form can be found at *ou:requirement*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A name for the requirement.	
:text	*str*	The text of the stated requirement.	Display: {'hint': 'text'}
:optional	*bool*	Set to true if the requirement is optional.	
:priority	*meta:priority*	The priority of the requirement.	
:goal	*ou:goal*	The goal that the requirement is designed to achieve.	
:active	*bool*	Set to true if the requirement is currently active.	
:issued	*time*	The time that the requirement was first issued.	
:period	*ival*	The time window where the goal must be met. Can be ongoing.	
:issuer	*ps:contact*	The contact information of the entity which issued the requirement.	
:assignee	*ps:contact*	The contact information of the entity which is assigned to meet the requirement.	
:deps	*array* type: *ou:requirement* sorted: True uniq: True	A list of sub-requirements which must be met to complete the requirement.	
:deps:min	*int* min: 0	The minimum number dependant requirements which must be met. If unset, assume all must be met.	

Source Edges:

source	verb	target	doc
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(meets)>	ou:requirement	The requirement is met by the source node.
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the targe
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the tar
it:exec:query	-(found)>	*	The target node was returned as a result of running the
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the tar
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target n
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target no
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target node

ou:suborg

Any parent/child relationship between two orgs. May represent ownership, organizational structure, etc.

The base type for the form can be found at *ou:suborg*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org which owns the sub organization.	Read Only: True
:sub	*ou:org*	The sub org which owned by the org.	Read Only: True
:perc	*int* min: 0 max: 100	The optional percentage of sub which is owned by org.	
:founded	*time*	The date on which the suborg relationship was founded.	
:dissolved	*time*	The date on which the suborg relationship was dissolved.	
:current	*bool*	Bool indicating if the suborg relationship still current.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 368 – continued from previous page

source	verb	target	doc
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:team

A GUID for a team within an organization.

The base type for the form can be found at *ou:team*.

Properties:

name	type	doc
:org	*ou:org*	A GUID for a human organization such as a company or military unit.
:name	*ou:name*	The name of an organization. This may be a formal name or informal name of the organization.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:technique

A specific technique used to achieve a goal.

The base type for the form can be found at *ou:technique*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The normalized name of the technique.	
:type	*ou:technique:taxonomy*	The taxonomy classification of the technique.	
:sophistication	*meta:sophistication*	The assessed sophistication of the technique.	
:desc	*str*	A description of the technique.	Display: {'hint': 'text'}
:tag	*syn:tag*	The tag used to annotate nodes where the technique was employed.	
:mitre:attack:tech	*it:mitre:attack:techniqu*	A mapping to a MITRE ATT&CK technique if applicable.	
:reporter	*ou:org*	The organization reporting on the technique.	
:reporter:name	*ou:name*	The name of the organization reporting on the technique.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target n
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target
it:exec:query	-(found)>	*	The target node was returned as a result of running the qu
it:prod:soft	-(uses)>	ou:technique	The software uses the technique.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target

continues on n

Table 370 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	ou:technique	The campaign used the technique.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target nod
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	ou:technique	The org uses the technique.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	ou:technique	The attacker used the technique in the attack.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the co
risk:compromise	-(uses)>	ou:technique	The attacker used the technique in the compromise.
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the ta
risk:extortion	-(uses)>	ou:technique	The attacker used the technique to extort the victim.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:mitigation	-(addresses)>	ou:technique	The mitigation addresses the technique.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	ou:technique	The threat cluster uses the technique.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	ou:technique	The tool uses the technique.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:technique:taxonomy

An analyst defined taxonomy to classify techniques in different disciplines.

The base type for the form can be found at *ou:technique:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ou:technique:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 371 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ou:user

A user name within an organization.

The base type for the form can be found at *ou:user*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org guid which owns the netblock.	Read Only: True
:user	*inet:user*	The username associated with the organization.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 372 – continued from previous page

source	verb	target	doc
`ou:contribution`	`-(includes)>`	*	The contribution includes the specific node.
`ou:org`	`-(has)>`	*	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	*	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

ou:vitals

Vital statistics about an org for a given time period.

The base type for the form can be found at *ou:vitals*.

Properties:

name	type	doc
`:asof`	*time*	The time that the vitals represent.
`:org`	*ou:org*	The resolved org.
`:orgname`	*ou:name*	The org name as reported by the source of the vitals.
`:orgfqdn`	*inet:fqdn*	The org FQDN as reported by the source of the vitals.
`:currency`	*econ:currency*	The currency of the econ:price values.
`:costs`	*econ:price*	The costs/expenditures over the period.
`:revenue`	*econ:price*	The gross revenue over the period.
`:profit`	*econ:price*	The net profit over the period.
`:valuation`	*econ:price*	The assessed value of the org.
`:shares`	*int*	The number of shares outstanding.
`:population`	*int*	The population of the org.
`:delta:costs`	*econ:price*	The change in costs over last period.
`:delta:revenue`	*econ:price*	The change in revenue over last period.
`:delta:profit`	*econ:price*	The change in profit over last period.
`:delta:valuation`	*econ:price*	The change in valuation over last period.
`:delta:population`	*int*	The change in population over last period.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:candidate

A candidate for office in a specific race.

The base type for the form can be found at *pol:candidate*.

Properties:

name	type	doc
:contact	*ps:contact*	The contact information of the candidate.
:race	*pol:race*	The race the candidate is participating in.
:campaign	*ou:campaign*	The official campaign to elect the candidate.
:winner	*bool*	Records the outcome of the race.
:party	*ou:org*	The declared political party of the candidate.
:incumbent	*bool*	Set to true if the candidate is an incumbent in this race.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 374 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:country

A GUID for a country.

The base type for the form can be found at *pol:country*.

Properties:

name	type	doc	opts
:flag	*file:bytes*	A thumbnail image of the flag of the country.	
:iso2	*pol:iso2*	The 2 digit ISO 3166 country code.	
:iso3	*pol:iso3*	The 3 digit ISO 3166 country code.	
:isonum	*pol:isonum*	The ISO integer country code.	
:pop	*int*	Deprecated. Please use :vitals::population.	Deprecated: True
:tld	*inet:fqdn*	A Fully Qualified Domain Name (FQDN).	
:name	*geo:name*	The name of the country.	
:names	*array* type: *geo:name* uniq: True sorted: True	An array of alternate or localized names for the country.	
:government	*ou:org*	The ou:org node which represents the government of the country.	
:place	*geo:place*	A geo:place node representing the geospatial properties of the country.	
:founded	*time*	The date that the country was founded.	
:dissolved	*time*	The date that the country was dissolved.	
:vitals	*pol:vitals*	The most recent known vitals for the country.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:election

An election involving one or more races for office.

The base type for the form can be found at *pol:election*.

Properties:

name	type	doc	opts
:name	*str* onespace: True lower: True	The name of the election.	Example: 2022 united states congressional midterm election
:time	*time*	The date of the election.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:immigration:status

A node which tracks the immigration status of a contact.

The base type for the form can be found at *pol:immigration:status*.

Properties:

name	type	doc	opts
:contact	*ps:contact*	The contact information for the immigration status record.	
:country	*pol:country*	The country that the contact is/has immigrated to.	
:type	*pol:immigration:status:*	A taxonomy entry for the immigration status type.	Example: citizen.naturalized
:state	*str* enums: requested, active,rejected, revoked, renounced	The state of the immigration status.	
:began	*time*	The time when the status was granted to the contact.	
:ended	*time*	The time when the status no longer applied to the contact.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 377 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:immigration:status:type:taxonomy

A taxonomy of immigration types.

The base type for the form can be found at *pol:immigration:status:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*pol:immigration:status:*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:office

An elected or appointed office.

The base type for the form can be found at *pol:office*.

Properties:

name	type	doc	opts
:title	*ou:jobtitl*	The title of the political office.	Example: united states senator
:positio	*ou:positi*	The position this office holds in the org chart for the governing body.	
:termlim	*int*	The maximum number of times a single person may hold the office.	
:govbody	*ou:org*	The governmental body which contains the office.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 379 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:pollingplace

An official place where ballots may be cast for a specific election.

The base type for the form can be found at *pol:pollingplace*.

Properties:

name	type	doc
:electic	*pol:electic*	The election that the polling place is designated for.
:name	*geo:name*	The name of the polling place at the time of the election. This may differ from the official place name.
:place	*geo:place*	The place where votes were cast.
:opens	*time*	The time that the polling place is scheduled to open.
:closes	*time*	The time that the polling place is scheduled to close.
:opened	*time*	The time that the polling place opened.
:closed	*time*	The time that the polling place closed.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 380 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:race

An individual race for office.

The base type for the form can be found at *pol:race*.

Properties:

name	type	doc
:election	*pol:election*	The election that includes the race.
:office	*pol:office*	The political office that the candidates in the race are running for.
:voters	*int*	The number of eligible voters for this race.
:turnout	*int*	The number of individuals who voted in this race.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:term

A term in office held by a specific individual.

The base type for the form can be found at *pol:term*.

Properties:

name	type	doc
:office	*pol:office*	The office held for the term.
:start	*time*	The start of the term of office.
:end	*time*	The end of the term of office.
:race	*pol:race*	The race that determined who held office during the term.
:contact	*ps:contact*	The contact information of the person who held office during the term.
:party	*ou:org*	The political party of the person who held office during the term.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

pol:vitals

A set of vital statistics about a country.

The base type for the form can be found at *pol:vitals*.

Properties:

name	type	doc
:country	*pol:country*	The country that the statistics are about.
:asof	*time*	The time that the vitals were measured.
:area	*geo:area*	The area of the country.
:population	*int*	The total number of people living in the country.
:currency	*econ:currency*	The national currency.
:econ:currency	*econ:currency*	The currency used to record price properties.
:econ:gdp	*econ:price*	The gross domestic product of the country.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.

continues on next page

Table 383 – continued from previous page

source	verb	target	doc
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:attachment

A file attachment added to a ticket or comment.

The base type for the form can be found at *proj:attachment*.

Properties:

name	type	doc
:name	*file:base*	The name of the file that was attached.
:file	*file:bytes*	The file that was attached.
:creator	*syn:user*	The synapse user who added the attachment.
:created	*time*	The time the attachment was added.
:ticket	*proj:ticket*	The ticket the attachment was added to.
:comment	*proj:comment*	The comment the attachment was added to.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 384 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:comment

A user comment on a ticket.

The base type for the form can be found at *proj:comment*.

Properties:

name	type	doc
:creator	*syn:user*	The synapse user who added the comment.
:created	*time*	The time the comment was added.
:updated	*time* ismax: `True`	The last time the comment was updated.
:ticket	*proj:ticket*	The ticket the comment was added to.
:text	*str*	The text of the comment.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:epic

A collection of tickets related to a topic.

The base type for the form can be found at *proj:epic*.

Properties:

name	type	doc
:name	*str* onespace: True	The name of the epic.
:project	*proj:project*	The project containing the epic.
:creator	*syn:user*	The synapse user who created the epic.
:created	*time*	The time the epic was created.
:updated	*time* ismax: True	The last time the epic was updated.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next page

Table 386 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:project

A project in a ticketing system.

The base type for the form can be found at *proj:project*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The project name.	
:type	*proj:project:type:taxon*	The project type.	
:desc	*str*	The project description.	Display: {'hint': 'text'}
:creator	*syn:user*	The synapse user who created the project.	
:created	*time*	The time the project was created.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:project:type:taxonomy

A type taxonomy for projects.

The base type for the form can be found at *proj:project:type:taxonomy*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:sprint

A timeboxed period to complete a set amount of work.

The base type for the form can be found at *proj:sprint*.

Properties:

name	type	doc
:name	*str* lower: True onespace: True	The name of the sprint.
:status	*str* enums: planned,current,completed	The sprint status.
:project	*proj:project*	The project containing the sprint.
:creator	*syn:user*	The synapse user who created the sprint.
:created	*time*	The date the sprint was created.
:period	*ival*	The interval for the sprint.
:desc	*str*	A description of the sprint.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 389 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

proj:ticket

A ticket in a ticketing system.

The base type for the form can be found at *proj:ticket*.

Properties:

name	type	doc
:project	*proj:project*	The project containing the ticket.
:ext:id	*str* strip: `True`	A ticket ID from an external system.
:ext:url	*inet:url*	A URL to the ticket in an external system.
:ext:creator	*ps:contact*	Ticket creator contact information from an external system.
:ext:assignee	*ps:contact*	Ticket assignee contact information from an external system.
:epic	*proj:epic*	The epic that includes the ticket.
:created	*time*	The time the ticket was created.
:updated	*time* ismax: `True`	The last time the ticket was updated.
:name	*str* onespace: `True`	The name of the ticket.
:desc	*str*	A description of the ticket.
:points	*int*	Optional SCRUM style story points value.
:status	*int* enums: `((0, 'new'), (10, 'in validation'), (20, 'in backlog'), (30, 'in sprint'), (40, 'in progress'), (50, 'in review'), (60, 'completed'), (70, 'done'), (80, 'blocked'))`	The ticket completion status.
:sprint	*proj:sprint*	The sprint that contains the ticket.
:priority	*int* enums: `((0, 'none'), (10, 'lowest'), (20, 'low'), (30, 'medium'), (40, 'high'), (50, 'highest'))`	The priority of the ticket.
:type	*str*	The type of ticket. (eg story / bug).

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:achievement

An instance of an individual receiving an award.

The base type for the form can be found at *ps:achievement*.

Properties:

name	type	doc
:awardee	*ps:contact*	The recipient of the award.
:award	*ou:award*	The award bestowed on the awardee.
:awarded	*time*	The date the award was granted to the awardee.
:expires	*time*	The date the award or certification expires.
:revoked	*time*	The date the award was revoked by the org.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 391 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:contact

A GUID for a contact info record.

The base type for the form can be found at *ps:contact*.

Properties:

name	type	doc	opts
:org	*ou:org*	The org which this contact represents.	
:type	*ps:contact:type:taxonoi*	The type of contact which may be used for entity resolution.	
:asof	*time*	A date/time value.	date: `The time this contact was created or modified.`
:person	*ps:person*	The ps:person GUID which owns this contact.	
:vitals	*ps:vitals*	The most recent known vitals for the contact.	
:name	*ps:name*	The person name listed for the contact.	
:desc	*str*	A description of this contact.	
:title	*ou:jobtitle*	The job/org title listed for this contact.	
:photo	*file:bytes*	The photo listed for this contact.	
:orgname	*ou:name*	The listed org/company name for this contact.	
:orgfqdn	*inet:fqdn*	The listed org/company FQDN for this contact.	
:user	*inet:user*	The username or handle for this contact.	

continues on next page

Table 392 – continued from previous page

name	type	doc	opts
:web:acct	*inet:web:acct*	The social media account for this contact.	
:web:group	*inet:web:group*	A web group representing this contact.	
:birth:place	*geo:place*	A fully resolved place of birth for this contact.	
:birth:place:loc	*loc*	The loc of the place of birth of this contact.	
:birth:place:name	*geo:name*	The name of the place of birth of this contact.	
:death:place	*geo:place*	A fully resolved place of death for this contact.	
:death:place:loc	*loc*	The loc of the place of death of this contact.	
:death:place:name	*geo:name*	The name of the place of death of this contact.	
:dob	*time*	The date of birth for this contact.	
:dod	*time*	The date of death for this contact.	
:url	*inet:url*	The home or main site for this contact.	
:email	*inet:email*	The main email address for this contact.	
:email:work	*inet:email*	The work email address for this contact.	
:loc	*loc*	Best known contact geopolitical location.	
:address	*geo:address*	The street address listed for the contact.	Display: {'hint': 'text'}
:place	*geo:place*	The place associated with this contact.	
:place:name	*geo:name*	The reported name of the place associated with this contact.	
:phone	*tel:phone*	The main phone number for this contact.	
:phone:fax	*tel:phone*	The fax number for this contact.	
:phone:work	*tel:phone*	The work phone number for this contact.	
:id:number	*ou:id:number*	An ID number issued by an org and associated with this contact.	

continues on next page

Table 392 – continued from previous page

name	type	doc	opts
:adid	*it:adid*	A Advertising ID associated with this contact.	
:imid	*tel:mob:imid*	An IMID associated with the contact.	
:imid:imei	*tel:mob:imei*	An IMEI associated with the contact.	
:imid:imsi	*tel:mob:imsi*	An IMSI associated with the contact.	
:names	*array* type: *ps:name* uniq: True sorted: True	An array of associated names/aliases for the person.	
:orgnames	*array* type: *ou:name* uniq: True sorted: True	An array of associated names/aliases for the organization.	
:emails	*array* type: *inet:email* uniq: True sorted: True	An array of secondary/associated email addresses.	
:web:accts	*array* type: *inet:web:acct* uniq: True sorted: True	An array of secondary/associated web accounts.	
:id:numbers	*array* type: *ou:id:number* uniq: True sorted: True	An array of secondary/associated IDs.	
:users	*array* type: *inet:user* uniq: True sorted: True	An array of secondary/associated user names.	
:crypto:address	*crypto:currency:addres*	A crypto currency address associated with the contact.	

continues on next page

Table 392 – continued from previous page

name	type	doc	opts
:lang	*lang:language*	The language specified for the contact.	
:langs	*array* type: *lang:language*	An array of alternative languages specified for the contact.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
ps:contac	-(has)>	*	The contact is or was in possession of the target node.
ps:contac	-(owns)>	*	The contact owns or owned the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:contact:type:taxonomy

A taxonomy of contact types.

The base type for the form can be found at *ps:contact:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*ps:contact:type:taxonoı*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 393 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:contactlist

A GUID for a list of associated contacts.

The base type for the form can be found at *ps:contactlist*.

Properties:

name	type	doc
:contacts	*array* type: *ps:contact* uniq: True split: , sorted: True	The array of contacts contained in the list.
:source:host	*it:host*	The host from which the contact list was extracted.
:source:file	*file:bytes*	The file from which the contact list was extracted.
:source:acct	*inet:web:acct*	The web account from which the contact list was extracted.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:education

A period of education for an individual.

The base type for the form can be found at *ps:education*.

Properties:

name	type	doc
:student	*ps:contact*	The contact of the person being educated.
:institution	*ps:contact*	The contact info for the org providing educational services.
:attended:first	*time*	The first date the student attended a class.
:attended:last	*time*	The last date the student attended a class.
:classes	*array* type: *edu:class* uniq: True sorted: True	The classes attended by the student.
:achievement	*ps:achievement*	The achievement awarded to the individual.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next page

Table 395 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:name

An arbitrary, lower spaced string with normalized whitespace.

The base type for the form can be found at *ps:name*.

An example of `ps:name`:

- `robert grey`

Properties:

name	type	doc
:sur	*ps:tokn*	The surname part of the name.
:middle	*ps:tokn*	The middle name part of the name.
:given	*ps:tokn*	The given name part of the name.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:person

A GUID for a person.

The base type for the form can be found at *ps:person*.

Properties:

name	type	doc	opts
:dob	*time*	The date on which the person was born.	
:dod	*time*	The date on which the person died.	
:img	*file:bytes*	Deprecated: use ps:person:photo.	Deprecated: True
:photo	*file:bytes*	The primary image of a person.	
:nick	*inet:user*	A username commonly used by the person.	
:vitals	*ps:vitals*	The most recent known vitals for the person.	
:name	*ps:name*	The localized name for the person.	
:name:sur	*ps:tokn*	The surname of the person.	
:name:middle	*ps:tokn*	The middle name of the person.	
:name:given	*ps:tokn*	The given name of the person.	
:names	*array* type: *ps:name* uniq: True sorted: True	Variations of the name for the person.	
:nicks	*array* type: *inet:user* uniq: True sorted: True	Usernames used by the person.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremer	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat):	geo:telem	The source node was seen at the geo:telem node place and time.
ps:persoı	-(has)>	*	The person is or was in possession of the target node.
ps:persoı	-(owns)>	*	The person owns or owned the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:person:has

A person owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time.

The base type for the form can be found at *ps:person:has*.

Properties:

name	type	doc	opts
:person	*ps:persoi*	The person who owns or controls the object or resource.	Read Only: True
:node	*ndef*	The object or resource that is owned or controlled by the person.	Read Only: True
:node:for	*str*	The form of the object or resource that is owned or controlled by the person.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next pag

Table 397 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:persona

A GUID for a suspected person.

The base type for the form can be found at *ps:persona*.

Properties:

name	type	doc
:person	*ps:person*	The real person behind the persona.
:dob	*time*	The Date of Birth (DOB) if known.
:img	*file:bytes*	The primary image of a suspected person.
:nick	*inet:user*	A username commonly used by the suspected person.
:name	*ps:name*	The localized name for the suspected person.
:name:sur	*ps:tokn*	The surname of the suspected person.
:name:middle	*ps:tokn*	The middle name of the suspected person.
:name:given	*ps:tokn*	The given name of the suspected person.
:names	*array* type: *ps:name* uniq: True sorted: True	Variations of the name for a persona.
:nicks	*array* type: *inet:user* uniq: True sorted: True	Usernames used by the persona.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:persona:has

A persona owns, controls, or has exclusive use of an object or resource, potentially during a specific period of time.

The base type for the form can be found at *ps:persona:has*.

Properties:

name	type	doc	opts
:persona	*ps:person*	The persona who owns or controls the object or resource.	Read Only: True
:node	*ndef*	The object or resource that is owned or controlled by the persona.	Read Only: True
:node:fo1	*str*	The form of the object or resource that is owned or controlled by the persona.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise

continues on next page

Table 399 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:proficiency

The assessment that a given contact possesses a specific skill.

The base type for the form can be found at *ps:proficiency*.

Properties:

name	type	doc
:skill	*ps:skill*	The skill in which the contact is proficient.
:contact	*ps:contact*	The contact which is proficient in the skill.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.

continues on next page

Table 400 – continued from previous page

source	verb	target	doc
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:skill

A specific skill which a person or organization may have.

The base type for the form can be found at *ps:skill*.

Properties:

name	type	doc
:name	*str* lower: True onespace: True	The name of the skill.
:type	*ps:skill:type:taxonomy*	The type of skill as a taxonomy.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:skill:type:taxonomy

A taxonomy of skill types.

The base type for the form can be found at *ps:skill:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: `True` Display: `{'hint': 'text'}`
:desc	*str*	A definition of the taxonomy entry.	Display: `{'hint': 'text'}`
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: `True`
:depth	*int*	The depth indexed from 0.	Read Only: `True`
:parent	*ps:skill:type:taxonomy*	The taxonomy parent.	Read Only: `True`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 402 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:tokn

A single name element (potentially given or sur).

The base type for the form can be found at *ps:tokn*.

An example of `ps:tokn`:

- `robert`

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.

continues on next

Table 403 – continued from previous page

source	verb	target	doc
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:vitals

Statistics and demographic data about a person or contact.

The base type for the form can be found at *ps:vitals*.

Properties:

name	type	doc
:asof	*time*	The time the vitals were gathered or computed.
:contact	*ps:contact*	The contact that the vitals are about.
:person	*ps:person*	The person that the vitals are about.
:height	*geo:dist*	The height of the person or contact.
:weight	*mass*	The weight of the person or contact.
:econ:currency	*econ:currency*	The currency that the price values are recorded using.
:econ:net:worth	*econ:price*	The net worth of the contact.
:econ:annual:income	*econ:price*	The yearly income of the contact.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

ps:workhist

A GUID representing entry in a contact's work history.

The base type for the form can be found at *ps:workhist*.

Properties:

name	type	doc
:contact	*ps:contact*	The contact which has the work history.
:org	*ou:org*	The org that this work history orgname refers to.
:orgname	*ou:name*	The reported name of the org the contact worked for.
:orgfqdn	*inet:fqdn*	The reported fqdn of the org the contact worked for.
:jobtype	*ou:jobtype*	The type of job.
:employment	*ou:employment*	The type of employment.
:jobtitle	*ou:jobtitle*	The job title.
:started	*time*	The date that the contact began working.
:ended	*time*	The date that the contact stopped working.
:duration	*duration*	The duration of the period of work.
:pay	*econ:price*	The estimated/average yearly pay for the work.
:currency	*econ:currency*	The currency that the yearly pay was delivered in.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node

continues on next page

Table 405 – continued from previous page

source	verb	target	doc
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:alert

An instance of an alert which indicates the presence of a risk.

The base type for the form can be found at *risk:alert*.

Properties:

name	type	doc	opts
:type	*risk:alert:taxonomy*	A type for the alert, as a taxonomy entry.	
:name	*str*	A brief name for the alert.	
:desc	*str*	A free-form description / overview of the alert.	Display: {'hint': 'text'}
:status	*int* enums: ((0, 'new'), (10, 'enrichment'), (20, 'todo'), (30, 'analysis'), (40, 'remediation'), (50, 'done'))	The status of the alert.	
:benign	*bool*	Set to true if the alert has been confirmed benign. Set to false if malicious.	
:priority	*meta:priority*	A priority rank for the alert.	
:severity	*meta:severity*	A severity rank for the alert.	
:verdict	*risk:alert:verdict:taxon*	A verdict about why the alert is malicious or benign, as a taxonomy entry.	Example: benign.false_positive
:assignee	*syn:user*	The Synapse user who is assigned to investigate the alert.	
:ext:assignee	*ps:contact*	The alert assignee contact information from an external system.	
:engine	*it:prod:softver*	The software that generated the alert.	
:detected	*time*	The time the alerted condition was detected.	
:vuln	*risk:vuln*	The optional vulnerability that the alert indicates.	
:attack	*risk:attack*	A confirmed attack that this alert indicates.	
:url	*inet:url*	A URL which documents the alert.	
:ext:id	*str*	An external identifier for the alert.	
:host	*it:host*	The host which generated the alert.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:alert:taxonomy

A taxonomy of alert types.

The base type for the form can be found at *risk:alert:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:alert:taxonomy*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 407 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:alert:verdict:taxonomy

A taxonomy of verdicts for the origin and validity of the alert.

The base type for the form can be found at *risk:alert:verdict:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:alert:verdict:taxon*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:attack

An instance of an actor attacking a target.

The base type for the form can be found at *risk:attack*.

Properties:

name	type	doc	opts
:desc	*str*	A description of the attack.	Display: {'hint': 'text'}
:type	*risk:attacktype*	A type for the attack, as a taxonomy entry.	Example: cno.phishing
:reporter	*ou:org*	The organization reporting on the attack.	
:reporter:name	*ou:name*	The name of the organization reporting on the attack.	
:time	*time*	Set if the time of the attack is known.	
:detected	*time*	The first confirmed detection time of the attack.	
:success	*bool*	Set if the attack was known to have succeeded or not.	
:targeted	*bool*	Set if the attack was assessed to be targeted or not.	
:goal	*ou:goal*	The tactical goal of this specific attack.	
:campaign	*ou:campaign*	Set if the attack was part of a larger campaign.	
:compromise	*risk:compromise*	A compromise that this attack contributed to.	
:severity	*meta:severity*	A severity rank for the attack.	
:sophistication	*meta:sophistication*	The assessed sophistication of the attack.	
:prev	*risk:attack*	The previous/parent attack in a list or hierarchy.	
:actor:org	*ou:org*	Deprecated. Please use :attacker to allow entity resolution.	Deprecated: True
:actor:person	*ps:person*	Deprecated. Please use :attacker to allow entity resolution.	Deprecated: True
:attacker	*ps:contact*	Contact information representing the attacker.	

continues on next page

Table 409 – continued from previous page

name	type	doc	opts
:target	*ps:contact*	Deprecated. Please use -(targets)> light weight edges.	Deprecated: True
:target:org	*ou:org*	Deprecated. Please use -(targets)> light weight edges.	Deprecated: True
:target:host	*it:host*	Deprecated. Please use -(targets)> light weight edges.	Deprecated: True
:target:person	*ps:person*	Deprecated. Please use -(targets)> light weight edges.	Deprecated: True
:target:place	*geo:place*	Deprecated. Please use -(targets)> light weight edges.	Deprecated: True
:via:ipv4	*inet:ipv4*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:via:ipv6	*inet:ipv6*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:via:email	*inet:email*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:via:phone	*tel:phone*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:vuln	*risk:vuln*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:url	*inet:url*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:host	*it:host*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:email	*inet:email*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:file	*file:bytes*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:server	*inet:server*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True
:used:software	*it:prod:softver*	Deprecated. Please use -(uses)> light weight edges.	Deprecated: True

continues on next page

Table 409 – continued from previous page

name	type	doc	opts
:techniques	*array* type: *ou:technique* sorted: True uniq: True	Deprecated for scalability. Please use -(uses)> ou:technique.	Deprecated: True
:url	*inet:url*	A URL which documents the attack.	
:ext:id	*str*	An external unique ID for the attack.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
risk:attac	-(targets)	ou:industry	The attack targeted the industry.
risk:attac	-(targets)	*	The attack targeted the target node.
risk:attac	-(uses)>	ou:technique	The attacker used the technique in the attack.
risk:attac	-(uses)>	risk:vuln	The attack used the vulnerability.
risk:attac	-(uses)>	*	The attack used the target node to facilitate the attack.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:attacktype

A taxonomy of attack types.

The base type for the form can be found at *risk:attacktype*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:attacktype*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 410 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:availability

A taxonomy of availability status values.

The base type for the form can be found at *risk:availability*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:availability*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:compromise

An instance of a compromise and its aggregate impact.

The base type for the form can be found at *risk:compromise*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A brief name for the compromise event.	
:desc	*str*	A prose description of the compromise event.	Display: {'hint': 'text'}
:reporter	*ou:org*	The organization reporting on the compromise.	
:reporter:name	*ou:name*	The name of the organization reporting on the compromise.	
:ext:id	*str*	An external unique ID for the compromise.	
:url	*inet:url*	A URL which documents the compromise.	
:type	*risk:compromisetype*	A type for the compromise, as a taxonomy entry.	Example: cno.breach
:vector	*risk:attack*	The attack assessed to be the initial compromise vector.	
:target	*ps:contact*	Contact information representing the target.	
:attacker	*ps:contact*	Contact information representing the attacker.	
:campaign	*ou:campaign*	The campaign that this compromise is part of.	
:time	*time*	Earliest known evidence of compromise.	
:lasttime	*time*	Last known evidence of compromise.	
:duration	*duration*	The duration of the compromise.	
:detected	*time*	The first confirmed detection time of the compromise.	
:loss:pii	*int*	The number of records compromised which contain PII.	
:loss:econ	*econ:price*	The total economic cost of the compromise.	
:loss:life	*int*	The total loss of life due to the compromise.	
:loss:bytes	*int*	An estimate of the volume of data compromised.	
:ransom:paid	*econ:price*	The value of the ransom paid by the target.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
risk:comprom:	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:comprom:	-(uses)>	ou:technique	The attacker used the technique in the compromise.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:compromisetype

A taxonomy of compromise types.

The base type for the form can be found at *risk:compromisetype*.

An example of `risk:compromisetype`:

- `cno.breach`

Properties:

name	type	doc	opts
`:title`	*str*	A brief title of the definition.	
`:summary`	*str*	Deprecated. Please use title/desc.	Deprecated: `True` Display: `{'hint': 'text'}`
`:desc`	*str*	A definition of the taxonomy entry.	Display: `{'hint': 'text'}`
`:sort`	*int*	A display sort order for siblings.	
`:base`	*taxon*	The base taxon.	Read Only: `True`
`:depth`	*int*	The depth indexed from 0.	Read Only: `True`
`:parent`	*risk:compromisetype*	The taxonomy parent.	Read Only: `True`

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.

continues on next

Table 413 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:extortion

An event where an attacker attempted to extort a victim.

The base type for the form can be found at *risk:extortion*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A name for the extortion event.	
:desc	*str*	A description of the extortion event.	Display: {'hint': 'text'}
:reporter	*ou:org*	The organization reporting on the extortion event.	
:reporter:name	*ou:name*	The name of the organization reporting on the extortion event.	
:demanded	*time*	The time that the attacker made their demands.	
:deadline	*time*	The time that the demand must be met.	
:goal	*ou:goal*	The goal of the attacker in extorting the victim.	
:type	*risk:extortion:type:taxo*	A type taxonomy for the extortion event.	
:attacker	*ps:contact*	The extortion attacker identity.	
:target	*ps:contact*	The extortion target identity.	
:success	*bool*	Set to true if the victim met the attacker's demands.	
:enacted	*bool*	Set to true if attacker carried out the threat.	
:public	*bool*	Set to true if the attacker publicly announced the extortion.	
:public:url	*inet:url*	The URL where the attacker publicly announced the extortion.	
:compromise	*risk:compromise*	The compromise which allowed the attacker to extort the target.	
:demanded:payment:	*econ:price*	The payment price which was demanded.	
:demanded:payment:	*econ:currency*	The currency in which payment was demanded.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirem	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
risk:extort	-(leveragec	*	The extortion event was based on attacker access to the target node.
risk:extort	-(uses)>	ou:techniqu	The attacker used the technique to extort the victim.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:extortion:type:taxonomy

A taxonomy of extortion event types.

The base type for the form can be found at *risk:extortion:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:extortion:type:taxc*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 415 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:hasvuln

Deprecated. Please use risk:vulnerable.

The base type for the form can be found at *risk:hasvuln*.

Properties:

name	type	doc
:vuln	*risk:vuln*	The vulnerability present in the target.
:person	*ps:person*	The vulnerable person.
:org	*ou:org*	The vulnerable org.
:place	*geo:place*	The vulnerable place.
:software	*it:prod:softver*	The vulnerable software.
:hardware	*it:prod:hardware*	The vulnerable hardware.
:spec	*mat:spec*	The vulnerable material specification.
:item	*mat:item*	The vulnerable material item.
:host	*it:host*	The vulnerable host.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:leak

An event where information was disclosed without permission.

The base type for the form can be found at *risk:leak*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	A simple name for the leak event.	
:desc	*str*	A description of the leak event.	Display: {'hint': 'text'}
:reporter	*ou:org*	The organization reporting on the leak event.	
:reporter:name	*ou:name*	The name of the organization reporting on the leak event.	
:disclosed	*time*	The time the leaked information was disclosed.	
:owner	*ps:contact*	The owner of the leaked information.	
:leaker	*ps:contact*	The identity which leaked the information.	
:type	*risk:leak:type:taxonom*	A type taxonomy for the leak.	
:goal	*ou:goal*	The goal of the leaker in disclosing the information.	
:compromise	*risk:compromise*	The compromise which allowed the leaker access to the information.	
:extortion	*risk:extortion*	The extortion event which used the threat of the leak as leverage.	
:public	*bool*	Set to true if the leaked information was made publicly available.	
:public:url	*inet:url*	The URL where the leaked information was made publicly available.	
:size:bytes	*int* min: 0	The approximate uncompressed size of the total data leaked.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremer	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat):	geo:telem	The source node was seen at the geo:telem node place and time.
risk:leal	-(leaked):	*	The leak included the disclosure of the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:leak:type:taxonomy

A taxonomy of leak event types.

The base type for the form can be found at *risk:leak:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:leak:type:taxonom*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 418 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:mitigation

A mitigation for a specific risk:vuln.

The base type for the form can be found at *risk:mitigation*.

Properties:

name	type	doc	opts
:vuln	*risk:vuln*	The vulnerability that this mitigation addresses.	
:name	*str* lower: True onespace: True	A brief name for this risk mitigation.	
:desc	*str*	A description of the mitigation approach for the vulnerability.	Display: {'hint': 'text'}
:software	*it:prod:softver*	A software version which implements a fix for the vulnerability.	
:hardware	*it:prod:hardware*	A hardware version which implements a fix for the vulnerability.	
:reporter	*ou:org*	The organization reporting on the mitigation.	
:reporter:name	*ou:name*	The name of the organization reporting on the mitigation.	
:mitre:attack:miti	*it:mitre:attack:mitigatic*	A mapping to a MITRE ATT&CK mitigation if applicable.	
:tag	*syn:tag*	The tag used to annotate nodes which have the mitigation in place.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
risk:mitigat	-(addresses	ou:technique	The mitigation addresses the technique.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:technique:masquerade

Represents the assessment that a node is designed to resemble another in order to mislead.

The base type for the form can be found at *risk:technique:masquerade*.

Properties:

name	type	doc
:node	*ndef*	The node masquerading as another.
:period	*ival*	The time period when the masquerading was active.
:target	*ndef*	The being masqueraded as.
:technique	*ou:technique*	The specific technique which describes the type of masquerading.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:threat

A threat cluster or subgraph of threat activity, as reported by a specific organization.

The base type for the form can be found at *risk:threat*.

Properties:

name	type	doc	opts
:name	*str* lower: `True` onespace: `True`	A brief descriptive name for the threat cluster.	Example: `apt1 (mandiant)`
:type	*risk:threat:type:taxonoi*	A type for the threat, as a taxonomy entry.	
:desc	*str*	A description of the threat cluster.	
:tag	*syn:tag*	The tag used to annotate nodes that are associated with the threat cluster.	
:active	*ival*	An interval for when the threat cluster is assessed to have been active.	
:reporter	*ou:org*	The organization reporting on the threat cluster.	
:reporter:name	*ou:name*	The name of the organization reporting on the threat cluster.	
:reporter:discover	*time*	The time that the reporting organization first discovered the threat cluster.	
:reporter:publishe	*time*	The time that the reporting organization first publicly disclosed the threat cluster.	
:org	*ou:org*	The authoritative organization for the threat cluster.	
:org:loc	*loc*	The reporting organization's assessed location of the threat cluster.	
:org:name	*ou:name*	The reporting organization's name for the threat cluster.	Example: `apt1`
:org:names	*array* type: *ou:name* sorted: `True` uniq: `True`	An array of alternate names for the threat cluster, according to the reporting organization.	
:country	*pol:country*	The reporting organization's assessed country of origin of the threat cluster.	
:country:code	*pol:iso2*	The 2 digit ISO 3166 country code for the threat cluster's assessed country of	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.
risk:threa	-(targets)	ou:industry	The threat cluster targets the industry.
risk:threa	-(targets)	*	The threat cluster targeted the target node.
risk:threa	-(uses)>	ou:technique	The threat cluster uses the technique.
risk:threa	-(uses)>	risk:vuln	The threat cluster uses the vulnerability.
risk:threa	-(uses)>	*	The threat cluster uses the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)	*	The purchase was used to acquire the target node.
it:app:snort:ru	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rul	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:tool:softw	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:threat:type:taxonomy

A taxonomy of threat types.

The base type for the form can be found at *risk:threat:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:threat:type:taxono*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 421 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:tool:software

A software tool used in threat activity, as reported by a specific organization.

The base type for the form can be found at *risk:tool:software*.

Properties:

name	type	doc	opts
:tag	*syn:tag*	The tag used to annotate nodes that are associated with the tool.	Example: `rep.mandiant.tabcteng`
:desc	*str*	A description of the tool.	
:type	*risk:tool:software:taxon*	A type for the tool, as a taxonomy entry.	
:used	*ival*	An interval for when the tool is assessed to have been deployed.	
:availability	*risk:availability*	The reporting organization's assessed availability of the tool.	
:sophistication	*meta:sophistication*	The reporting organization's assessed sophistication of the tool.	
:reporter	*ou:org*	The organization reporting on the tool.	
:reporter:name	*ou:name*	The name of the organization reporting on the tool.	
:reporter:discovered	*time*	The time that the reporting organization first discovered the tool.	
:reporter:published	*time*	The time that the reporting organization first publicly disclosed the tool.	
:soft	*it:prod:soft*	The authoritative software family for the tool.	
:soft:name	*it:prod:softname*	The reporting organization's name for the tool.	
:soft:names	*array* type: *it:prod:softname* uniq: `True` sorted: `True`	An array of alternate names for the tool, according to the reporting organization.	
:techniques	*array* type: *ou:technique* uniq: `True` sorted: `True`	Deprecated for scalability. Please use -(uses)> ou:technique.	Deprecated: `True`
:mitre:attack:software	*it:mitre:attack:software*	A mapping to a MITRE ATT&CK software if applicable.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
risk:tool:softw	-(uses)>	ou:technique	The tool uses the technique.
risk:tool:softw	-(uses)>	risk:vuln	The tool uses the vulnerability.
risk:tool:softw	-(uses)>	*	The tool uses the target node.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise.
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node.
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:tool:software:taxonomy

A taxonomy of software / tool types.

The base type for the form can be found at *risk:tool:software:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:tool:software:taxoı*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.

continues on next page

Table 423 – continued from previous page

source	verb	target	doc
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:vuln

A unique vulnerability.

The base type for the form can be found at *risk:vuln*.

Properties:

name	type	doc	opts
:name	*risk:vulnname*	A user specified name for the vulnerability.	
:names	*array* type: *risk:vulnname* sorted: True uniq: True	An array of alternate names for the vulnerability.	
:type	*risk:vuln:type:taxonom*	A taxonomy type entry for the vulnerability.	
:desc	*str*	A description of the vulnerability.	Display: {'hint': 'text'}
:severity	*meta:severity*	The severity of the vulnerability.	
:priority	*meta:priority*	The priority of the vulnerability.	
:reporter	*ou:org*	The organization reporting on the vulnerability.	

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:reporter:name	*ou:name*	The name of the organization reporting on the vulnerability.	
:mitigated	*bool*	Set to true if a mitigation/fix is available for the vulnerability.	
:exploited	*bool*	Set to true if the vulnerability has been exploited in the wild.	
:timeline:discover	*time* ismin: `True`	The earliest known discovery time for the vulnerability.	
:timeline:publishe	*time* ismin: `True`	The earliest known time the vulnerability was published.	
:timeline:vendor:n	*time* ismin: `True`	The earliest known vendor notification time for the vulnerability.	
:timeline:vendor:f	*time* ismin: `True`	The earliest known time the vendor issued a fix for the vulnerability.	
:timeline:exploite	*time* ismin: `True`	The earliest known time when the vulnerability was exploited in the wild.	
:id	*str* strip: `True`	An identifier for the vulnerability.	
:cve	*it:sec:cve*	The CVE ID of the vulnerability.	
:cve:desc	*str*	The description of the vulnerability according to the CVE database.	Display: `{'hint': 'text'}`
:cve:url	*inet:url*	A URL linking this vulnerability to the CVE description.	
:cve:references	*array* type: *inet:url* uniq: `True` sorted: `True`	An array of documentation URLs provided by the CVE database.	

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:nist:nvd:source	*ou:name*	The name of the organization which reported the vulnerability to NIST.	
:nist:nvd:publishe	*time*	The date the vulnerability was first published in the NVD.	
:nist:nvd:modified	*time* ismax: True	The date the vulnerability was last modified in the NVD.	
:cisa:kev:name	*str*	The name of the vulnerability according to the CISA KEV database.	
:cisa:kev:desc	*str*	The description of the vulnerability according to the CISA KEV database.	
:cisa:kev:action	*str*	The action to mitigate the vulnerability according to the CISA KEV database.	
:cisa:kev:vendor	*ou:name*	The vendor name listed in the CISA KEV database.	
:cisa:kev:product	*it:prod:softname*	The product name listed in the CISA KEV database.	
:cisa:kev:added	*time*	The date the vulnerability was added to the CISA KEV database.	
:cisa:kev:duedate	*time*	The date the action is due according to the CISA KEV database.	
:cvss:v2	*cvss:v2*	The CVSS v2 vector for the vulnerability.	
:cvss:v2_0:score	*float*	The CVSS v2.0 overall score for the vulnerability.	
:cvss:v2_0:score:b	*float*	The CVSS v2.0 base score for the vulnerability.	
:cvss:v2_0:score:t	*float*	The CVSS v2.0 temporal score for the vulnerability.	
:cvss:v2_0:score:e	*float*	The CVSS v2.0 environmental score for the vulnerability.	

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:cvss:v3	*cvss:v3*	The CVSS v3 vector for the vulnerability.	
:cvss:v3_0:score	*float*	The CVSS v3.0 overall score for the vulnerability.	
:cvss:v3_0:score:b	*float*	The CVSS v3.0 base score for the vulnerability.	
:cvss:v3_0:score:t	*float*	The CVSS v3.0 temporal score for the vulnerability.	
:cvss:v3_0:score:e	*float*	The CVSS v3.0 environmental score for the vulnerability.	
:cvss:v3_1:score	*float*	The CVSS v3.1 overall score for the vulnerability.	
:cvss:v3_1:score:b	*float*	The CVSS v3.1 base score for the vulnerability.	
:cvss:v3_1:score:t	*float*	The CVSS v3.1 temporal score for the vulnerability.	
:cvss:v3_1:score:e	*float*	The CVSS v3.1 environmental score for the vulnerability.	
:cvss:av	*str* enums: N,A,P,L	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:ac	*str* enums: L,H	Deprecated. Please use :cvss:v3.	Display: {'enums': (('Low', 'L'), ('High', 'H'))} Deprecated: True

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:cvss:pr	*str* enums: N,L,H	Deprecated. Please use :cvss:v3.	Display: {'enums': ({'title': 'None', 'value': 'N', 'doc': 'FIXME privs stuff'}, {'title': 'Low', 'value': 'L', 'doc': 'FIXME privs stuff'}, {'title': 'High', 'value': 'H', 'doc': 'FIXME privs stuff'})} Deprecated: True
:cvss:ui	*str* enums: N,R	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:s	*str* enums: U,C	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:c	*str* enums: N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:i	*str* enums: N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:a	*str* enums: N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:e	*str* enums: X,U,P,F,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:rl	*str* enums: X,O,T,W,U	Deprecated. Please use :cvss:v3.	Deprecated: True

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:cvss:rc	*str* enums: X,U,R,C	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mav	*str* enums: X,N,A,L,P	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mac	*str* enums: X,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mpr	*str* enums: X,N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mui	*str* enums: X,N,R	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:ms	*str* enums: X,U,C	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mc	*str* enums: X,N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:mi	*str* enums: X,N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:ma	*str* enums: X,N,L,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:cr	*str* enums: X,L,M,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:ir	*str* enums: X,L,M,H	Deprecated. Please use :cvss:v3.	Deprecated: True
:cvss:ar	*str* enums: X,L,M,H	Deprecated. Please use :cvss:v3.	Deprecated: True

continues on next page

Table 424 – continued from previous page

name	type	doc	opts
:cvss:score	*float*	Deprecated. Please use version specific score properties.	Deprecated: True
:cvss:score:base	*float*	Deprecated. Please use version specific score properties.	Deprecated: True
:cvss:score:tempor	*float*	Deprecated. Please use version specific score properties.	Deprecated: True
:cvss:score:enviro	*float*	Deprecated. Please use version specific score properties.	Deprecated: True
:cwes	*array* type: *it:sec:cwe* uniq: True sorted: True	An array of MITRE CWE values that apply to the vulnerability.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target noc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target noc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.

continues on next

Table 425 – continued from previous page

source	verb	target	doc
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	risk:vuln	The attack used the vulnerability.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compr
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	risk:vuln	The threat cluster uses the vulnerability.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	risk:vuln	The tool uses the vulnerability.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:vuln:soft:range

A contiguous range of software versions which contain a vulnerability.

The base type for the form can be found at *risk:vuln:soft:range*.

Properties:

name	type	doc
:vuln	*risk:vuln*	The vulnerability present in this software version range.
:version:min	*it:prod:softver*	The minimum version which is vulnerable in this range.
:version:max	*it:prod:softver*	The maximum version which is vulnerable in this range.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next pag

Table 426 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:vuln:type:taxonomy

A taxonomy of vulnerability types.

The base type for the form can be found at *risk:vuln:type:taxonomy*.

Properties:

name	type	doc	opts
:title	*str*	A brief title of the definition.	
:summary	*str*	Deprecated. Please use title/desc.	Deprecated: True Display: {'hint': 'text'}
:desc	*str*	A definition of the taxonomy entry.	Display: {'hint': 'text'}
:sort	*int*	A display sort order for siblings.	
:base	*taxon*	The base taxon.	Read Only: True
:depth	*int*	The depth indexed from 0.	Read Only: True
:parent	*risk:vuln:type:taxonom*	The taxonomy parent.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.

continues on next page

Table 427 – continued from previous page

source	verb	target	doc
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:vulnerable

Indicates that a node is susceptible to a vulnerability.

The base type for the form can be found at *risk:vulnerable*.

Properties:

name	type	doc
:vuln	*risk:vuln*	The vulnerability that the node is susceptible to.
:period	*ival*	The time window where the node was vulnerable.
:node	*ndef*	The node which is vulnerable.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 428 – continued from previous page

source	verb	target	doc
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

risk:vulnname

A vulnerability name such as log4j or rowhammer.

The base type for the form can be found at *risk:vulnname*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 429 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

rsa:key

An RSA keypair modulus and public exponent.

The base type for the form can be found at *rsa:key*.

Properties:

name	type	doc	opts
:mod	*hex*	The RSA key modulus.	Read Only: True
:pub:exp	*int*	The public exponent of the key.	Read Only: True
:bits	*int*	The length of the modulus in bits.	
:priv:exp	*hex*	The private exponent of the key.	
:priv:p	*hex*	One of the two private primes.	
:priv:q	*hex*	One of the two private primes.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:evidence

An assessment of how an observation supports or refutes a hypothesis.

The base type for the form can be found at *sci:evidence*.

Properties:

name	type	doc	opts
:hypothe:	*sci:experin*	The hypothesis which the evidence supports or refutes.	
:observa	*sci:observ*	The observation which supports or refutes the hypothesis.	
:summary	*str*	A summary of how the observation supports or refutes the hypothesis.	Display: {'hint': 'text'}
:refutes	*bool*	Set to true if the evidence refutes the hypothesis or false if it supports the hypothesis.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
sci:evidenc	-(has)>	*	The evidence includes observations from the target nodes.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:experiment

An instance of running an experiment.

The base type for the form can be found at *sci:experiment*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name of the experiment.	
:summary	*str*	A summary of the experiment.	Display: {'hint': 'text'}
:time	*time*	The time when the experiment was initiated.	
:type	*sci:experiment:type:tax*	The type of experiment as a user defined taxonomy.	
:window	*ival*	The time window where the experiment was run.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
sci:experime	-(uses)>	*	The experiment used the target nodes when it was run.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 432 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:experiment:type:taxonomy

A taxonomy of experiment types.

The base type for the form can be found at *sci:experiment:type:taxonomy*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.

continues on next page

Table 433 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:hypothesis

A hypothesis or theory.

The base type for the form can be found at *sci:hypothesis*.

Properties:

name	type	doc	opts
:name	*str* lower: True onespace: True	The name of the hypothesis.	
:type	*sci:hypothesis:type:taxo*	The type of hypothesis as a user defined taxonomy.	
:summary	*str*	A summary of the hypothesis.	Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:hypothesis:type:taxonomy

A taxonomy of hypothesis types.

The base type for the form can be found at *sci:hypothesis:type:taxonomy*.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

sci:observation

An observation which may have resulted from an experiment.

The base type for the form can be found at *sci:observation*.

Properties:

name	type	doc	opts
:experimen	*sci:experimei*	The experiment which produced the observation.	
:summary	*str*	A summary of the observation.	Display: {'hint': 'text'}
:time	*time*	The time that the observation occurred.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.
sci:observati	-(has)>	*	The observations are summarized from the target nodes.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.

continues on next page

Table 436 – continued from previous page

source	verb	target	doc
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.

syn:cmd

A Synapse storm command.

The base type for the form can be found at *syn:cmd*.

Properties:

name	type	doc	opts
:doc	*str* strip: True	Description of the command.	Display: {'hint': 'text'}
:package	*str* strip: True	Storm package which provided the command.	
:svciden	*guid* strip: True	Storm service iden which provided the package.	
:input	*array* type: *syn:form*	The list of forms accepted by the command as input.	uniq: True sorted: True Read Only: True
:output	*array* type: *syn:form*	The list of forms produced by the command as output.	uniq: True sorted: True Read Only: True
:nodedata	*array* type: *syn:nodedata*	The list of nodedata that may be added by the command.	uniq: True sorted: True Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.

continues on next page

Table 437 – continued from previous page

source	verb	target	doc
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:cron

A Cortex cron job.

The base type for the form can be found at *syn:cron*.

Properties:

name	type	doc	opts
:doc	*str*	A description of the cron job.	Display: {'hint': 'text'}
:name	*str*	A user friendly name/alias for the cron job.	
:storm	*str*	The storm query executed by the cron job.	Read Only: True Display: {'hint': 'text'}

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:form

A Synapse form used for representing nodes in the graph.

The base type for the form can be found at *syn:form*.

Properties:

name	type	doc	opts
:doc	*str* strip: True	The docstring for the form.	Read Only: True
:type	*syn:type*	Synapse type for this form.	Read Only: True
:runt	*bool*	Whether or not the form is runtime only.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.

continues on next page

Table 439 – continued from previous page

source	verb	target	doc
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:prop

A Synapse property.

The base type for the form can be found at *syn:prop*.

Properties:

name	type	doc	opts
:doc	*str* strip: True	Description of the property definition.	
:form	*syn:form*	The form of the property.	Read Only: True
:type	*syn:type*	The synapse type for this property.	Read Only: True
:relname	*str* strip: True	Relative property name.	Read Only: True
:univ	*bool*	Specifies if a prop is universal.	Read Only: True
:base	*str* strip: True	Base name of the property.	Read Only: True
:ro	*bool*	If the property is read-only after being set.	Read Only: True
:extmodel	*bool*	If the property is an extended model property or not.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:tag

The base type for a synapse tag.

The base type for the form can be found at *syn:tag*.

Properties:

name	type	doc	opts
:up	*syn:tag*	The parent tag for the tag.	Read Only: True
:isnow	*syn:tag*	Set to an updated tag if the tag has been renamed.	
:doc	*str*	A short definition for the tag.	Display: {'hint': 'text'}
:doc:url	*inet:url*	A URL link to additional documentation about the tag.	
:depth	*int*	How deep the tag is in the hierarchy.	Read Only: True
:title	*str*	A display title for the tag.	
:base	*str*	The tag base name. Eg baz for foo.bar.baz .	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.

continues on next page

Table 441 – continued from previous page

source	verb	target	doc
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:tagprop

A user defined tag property.

The base type for the form can be found at *syn:tagprop*.

Properties:

name	type	doc	opts
:doc	*str* strip: True	Description of the tagprop definition.	
:type	*syn:type*	The synapse type for this tagprop.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.

continues on next page

Table 442 – continued from previous page

source	verb	target	doc
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:trigger

A Cortex trigger.

The base type for the form can be found at *syn:trigger*.

Properties:

name	type	doc	opts
:vers	*int*	Trigger version.	Read Only: True
:doc	*str*	A documentation string describing the trigger.	Display: {'hint': 'text'}
:name	*str*	A user friendly name/alias for the trigger.	
:cond	*str* strip: True lower: True	The trigger condition.	Read Only: True
:user	*str*	User who owns the trigger.	Read Only: True
:storm	*str*	The Storm query for the trigger.	Read Only: True Display: {'hint': 'text'}
:enabled	*bool*	Trigger enabled status.	Read Only: True
:form	*str* lower: True strip: True	Form the trigger is watching for.	
:verb	*str* lower: True strip: True	Edge verb the trigger is watching for.	
:n2form	*str* lower: True strip: True	N2 form the trigger is watching for.	
:prop	*str* lower: True strip: True	Property the trigger is watching for.	
:tag	*str* lower: True strip: True	Tag the trigger is watching for.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

syn:type

A Synapse type used for normalizing nodes and properties.

The base type for the form can be found at *syn:type*.

Properties:

name	type	doc	opts
:doc	*str* strip: True	The docstring for the type.	Read Only: True
:ctor	*str* strip: True	The python ctor path for the type object.	Read Only: True
:subof	*syn:type*	Type which this inherits from.	Read Only: True
:opts	*data*	Arbitrary type options.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 444 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:call

A guid for a telephone call record.

The base type for the form can be found at *tel:call*.

Properties:

name	type	doc	opts
:src	*tel:phone*	The source phone number for a call.	
:dst	*tel:phone*	The destination phone number for a call.	
:time	*time*	The time the call was initiated.	
:duration	*int*	The duration of the call in seconds.	
:connected	*bool*	Indicator of whether the call was connected.	
:text	*str*	The text transcription of the call.	Display: {'hint': 'text'}
:file	*file:bytes*	A file containing related media.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:carrier

The fusion of a MCC/MNC.

The base type for the form can be found at *tel:mob:carrier*.

Properties:

name	type	doc	opts
:mcc	*tel:mob:mcc*	ITU Mobile Country Code.	Read Only: True
:mnc	*tel:mob:mnc*	ITU Mobile Network Code.	Read Only: True
:org	*ou:org*	Organization operating the carrier.	
:loc	*loc*	Location the carrier operates from.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:cell

A mobile cell site which a phone may connect to.

The base type for the form can be found at *tel:mob:cell*.

Properties:

name	type	doc	opts
:carrier	*tel:mob:carrier*	Mobile carrier.	Read Only: True
:carrier:mcc	*tel:mob:mcc*	Mobile Country Code.	Read Only: True
:carrier:mnc	*tel:mob:mnc*	Mobile Network Code.	Read Only: True
:lac	*int*	Location Area Code. LTE networks may call this a TAC.	Read Only: True
:cid	*int*	The Cell ID.	Read Only: True
:radio	*str* lower: 1 onespace: 1	Cell radio type.	
:latlong	*geo:latlong*	Last known location of the cell site.	
:loc	*loc*	Location at which the cell is operated.	
:place	*geo:place*	The place associated with the latlong property.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 447 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:imei

An International Mobile Equipment Id.

The base type for the form can be found at *tel:mob:imei*.

An example of `tel:mob:imei`:

- `490154203237518`

Properties:

name	type	doc	opts
:tac	*tel:mob:tac*	The Type Allocate Code within the IMEI.	Read Only: True
:serial	*int*	The serial number within the IMEI.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:imid

Fused knowledge of an IMEI/IMSI used together.

The base type for the form can be found at *tel:mob:imid*.

An example of `tel:mob:imid`:

- `(490154203237518, 310150123456789)`

Properties:

name	type	doc	opts
:imei	*tel:mob:imei*	The IMEI for the phone hardware.	Read Only: True
:imsi	*tel:mob:imsi*	The IMSI for the phone subscriber.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro

continues on next

Table 449 – continued from previous page

source	verb	target	doc
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:imsi

An International Mobile Subscriber Id.

The base type for the form can be found at *tel:mob:imsi*.

An example of `tel:mob:imsi`:

- `310150123456789`

Properties:

name	type	doc	opts
:mcc	*tel:mob:mcc*	The Mobile Country Code.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.

continues on next

Table 450 – continued from previous page

source	verb	target	doc
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:imsiphone

Fused knowledge of an IMSI assigned phone number.

The base type for the form can be found at *tel:mob:imsiphone*.

An example of `tel:mob:imsiphone`:

- `(310150123456789, "+7(495) 124-59-83")`

Properties:

name	type	doc	opts
:phone	*tel:phone*	The phone number assigned to the IMSI.	Read Only: True
:imsi	*tel:mob:imsi*	The IMSI with the assigned phone number.	Read Only: True

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:mcc

ITU Mobile Country Code.

The base type for the form can be found at *tel:mob:mcc*.

Properties:

name	type	doc
:loc	*loc*	Location assigned to the MCC.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:tac

A mobile Type Allocation Code.

The base type for the form can be found at *tel:mob:tac*.

An example of `tel:mob:tac`:

- `49015420`

Properties:

name	type	doc
`:org`	*ou:org*	The org guid for the manufacturer.
`:manu`	*str* lower: 1	The TAC manufacturer name.
`:model`	*str* lower: 1	The TAC model name.
`:internal`	*str* lower: 1	The TAC internal model name.

Source Edges:

source	verb	target	doc
`*`	`-(meets)>`	`ou:requireme`	The requirement is met by the source node.
`*`	`-(refs)>`	`*`	The source node contains a reference to the target node.
`*`	`-(seenat)`	`geo:telem`	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
`*`	`-(refs)>`	`*`	None
`econ:purchase`	`-(acquired)>`	`*`	The purchase was used to acquire the target node.
`it:app:snort:rule`	`-(detects)>`	`*`	The snort rule is intended for use in detecting the target node.
`it:app:yara:rule`	`-(detects)>`	`*`	The YARA rule is intended for use in detecting the target nod
`it:exec:query`	`-(found)>`	`*`	The target node was returned as a result of running the query.
`meta:note`	`-(about)>`	`*`	The meta:note is about the target node.
`meta:rule`	`-(detects)>`	`*`	The meta:rule is designed to detect instances of the target nod
`meta:rule`	`-(matches)>`	`*`	The meta:rule has matched on target node.
`meta:source`	`-(seen)>`	`*`	The meta:source observed the target node.

continues on next

Table 453 – continued from previous page

source	verb	target	doc
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:mob:telem

A single mobile telemetry measurement.

The base type for the form can be found at *tel:mob:telem*.

Properties:

name	type	doc
:time	*time*	A date/time value.
:latlong	*geo:latlong*	A Lat/Long string specifying a point on Earth.
:http:request	*inet:http:request*	The HTTP request that the telemetry was extracted from.
:host	*it:host*	The host that generated the mobile telemetry data.
:place	*geo:place*	The place representing the location of the mobile telemetry sample.
:loc	*loc*	The geo-political location of the mobile telemetry sample.
:accuracy	*geo:dist*	The reported accuracy of the latlong telemetry reading.
:cell	*tel:mob:cell*	A mobile cell site which a phone may connect to.
:cell:carrier	*tel:mob:carrier*	The fusion of a MCC/MNC.
:imsi	*tel:mob:imsi*	An International Mobile Subscriber Id.
:imei	*tel:mob:imei*	An International Mobile Equipment Id.
:phone	*tel:phone*	A phone number.
:mac	*inet:mac*	A 48-bit Media Access Control (MAC) address.
:ipv4	*inet:ipv4*	An IPv4 address.
:ipv6	*inet:ipv6*	An IPv6 address.
:wifi	*inet:wifi:ap*	An SSID/MAC address combination for a wireless access point.
:wifi:ssid	*inet:wifi:ssid*	A WiFi service set identifier (SSID) name.
:wifi:bssid	*inet:mac*	A 48-bit Media Access Control (MAC) address.
:adid	*it:adid*	An advertising identification string.
:aaid	*it:os:android:aaid*	An android advertising identification string.
:idfa	*it:os:ios:idfa*	An iOS advertising identification string.
:name	*ps:name*	An arbitrary, lower spaced string with normalized whitespace.
:email	*inet:email*	An e-mail address.
:acct	*inet:web:acct*	An account with a given Internet-based site or service.
:app	*it:prod:softver*	A specific version of a software product.
:data	*data*	Arbitrary json compatible data.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requirement	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.

continues on next page

Table 454 – continued from previous page

source	verb	target	doc
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:phone

A phone number.

The base type for the form can be found at *tel:phone*.

An example of `tel:phone`:

- `+15558675309`

Properties:

name	type	doc
:loc	*loc*	The location associated with the number.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

tel:txtmesg

A guid for an individual text message.

The base type for the form can be found at *tel:txtmesg*.

Properties:

name	type	doc	opts
:from	*tel:phone*	The phone number assigned to the sender.	
:to	*tel:phone*	The phone number assigned to the primary recipient.	
:recipients	*array* type: *tel:phone* uniq: True sorted: True	An array of phone numbers for additional recipients of the message.	
:svctype	*str* enums: sms,mms,rcs strip: 1 lower: 1	The message service type (sms, mms, rcs).	
:time	*time*	The time the message was sent.	
:text	*str*	The text of the message.	Display: {'hint': 'text'}
:file	*file:bytes*	A file containing related media.	

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 456 – continued from previous page

source	verb	target	doc
`ou:campaign`	`-(uses)>`	*	The campaign made use of the target node.
`ou:contribution`	`-(includes)>`	*	The contribution includes the specific node.
`ou:org`	`-(has)>`	*	The organization is or was in possession of the target node.
`ou:org`	`-(owns)>`	*	The organization owns or owned the target node.
`ou:org`	`-(targets)>`	*	The organization targets the target node.
`ou:org`	`-(uses)>`	*	The ou:org makes use of the target node.
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

transport:air:craft

An individual aircraft.

The base type for the form can be found at *transport:air:craft*.

Properties:

name	type	doc
:tailnum	*transport:air:tailnum*	The aircraft tail number.
:type	*str* lower: True strip: True	The type of aircraft.
:built	*time*	The date the aircraft was constructed.
:make	*str* lower: True strip: True	The make of the aircraft.
:model	*str* lower: True strip: True	The model of the aircraft.
:serial	*str* strip: True	The serial number of the aircraft.
:operator	*ps:contact*	Contact info representing the person or org that operates the aircraft.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.

continues on next page

Table 457 – continued from previous page

source	verb	target	doc
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:flight

An individual instance of a flight.

The base type for the form can be found at *transport:air:flight*.

Properties:

name	type	doc
:num	*transport:air:flightnum*	The flight number of this flight.
:scheduled:departure	*time*	The time this flight was originally scheduled to depart.
:scheduled:arrival	*time*	The time this flight was originally scheduled to arrive.
:departed	*time*	The time this flight departed.
:arrived	*time*	The time this flight arrived.
:carrier	*ou:org*	The org which operates the given flight number.
:craft	*transport:air:craft*	The aircraft that flew this flight.
:tailnum	*transport:air:tailnum*	The tail/registration number at the time the aircraft flew this flight.
:to:port	*transport:air:port*	The destination airport of this flight.
:from:port	*transport:air:port*	The origin airport of this flight.
:stops	*array* type: *transport:air:port*	An ordered list of airport codes for stops which occurred during this flight.
:cancelled	*bool*	Set to true for cancelled flights.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.

continues on next page

Table 458 – continued from previous page

source	verb	target	doc
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:flightnum

A commercial flight designator including airline and serial.

The base type for the form can be found at *transport:air:flightnum*.

An example of `transport:air:flightnum`:

- `ua2437`

Properties:

name	type	doc
:carrier	*ou:org*	The org which operates the given flight number.
:to:port	*transport:air:port*	The most recently registered destination for the flight number.
:from:port	*transport:air:port*	The most recently registered origin for the flight number.
:stops	*array* type: *transport:air:port*	An ordered list of aiport codes for the flight segments.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:occupant

An occupant of a specific flight.

The base type for the form can be found at *transport:air:occupant*.

Properties:

name	type	doc
:type	*str* lower: True	The type of occupant such as pilot, crew or passenger.
:flight	*transport:air:flight*	The flight that the occupant was aboard.
:seat	*str* lower: True	The seat assigned to the occupant.
:contact	*ps:contact*	The contact information of the occupant.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 460 – continued from previous page

source	verb	target	doc
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:port

An IATA assigned airport code.

The base type for the form can be found at *transport:air:port*.

Properties:

name	type	doc
:name	*str* lower: True onespace: True	The name of the airport.
:place	*geo:place*	The place where the IATA airport code is assigned.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.

continues on next page

Table 461 – continued from previous page

source	verb	target	doc
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:tailnum

An aircraft registration number or military aircraft serial number.

The base type for the form can be found at *transport:air:tailnum*.

An example of `transport:air:tailnum`:

- `ff023`

  Properties:

name	type	doc
:loc	*loc*	The geopolitical location that the tailnumber is allocated to.
:type	*str* lower: True strip: True	A type which may be specific to the country prefix.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requireme	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target nod
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target nod
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compro
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.

continues on next

Table 462 – continued from previous page

source	verb	target	doc
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:air:telem

A telemetry sample from an aircraft in transit.

The base type for the form can be found at *transport:air:telem*.

Properties:

name	type	doc
:flight	*transport:air:flight*	The flight being measured.
:latlong	*geo:latlong*	The lat/lon of the aircraft at the time.
:loc	*loc*	The location of the aircraft at the time.
:place	*geo:place*	The place that the lat/lon geocodes to.
:accuracy	*geo:dist*	The horizontal accuracy of the latlong sample.
:course	*transport:direction*	The direction, in degrees from true North, that the aircraft is traveling.
:heading	*transport:direction*	The direction, in degrees from true North, that the nose of the aircraft is pointed.
:speed	*velocity*	The ground speed of the aircraft at the time.
:airspeed	*velocity*	The air speed of the aircraft at the time.
:verticalspeed	*velocity* relative: True	The relative vertical speed of the aircraft at the time.
:altitude	*geo:altitude*	The altitude of the aircraft at the time.
:altitude:accuracy	*geo:dist*	The vertical accuracy of the altitude measurement.
:time	*time*	The time the telemetry sample was taken.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromis
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target nod
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:land:license

A license to operate a land vehicle issued to a contact.

The base type for the form can be found at *transport:land:license*.

Properties:

name	type	doc
:id	*str* strip: True	The license ID.
:contact	*ps:contact*	The contact info of the registrant.
:issued	*time*	The time the license was issued.
:expires	*time*	The time the license expires.
:issuer	*ou:org*	The org which issued the license.
:issuer:name	*ou:name*	The name of the org which issued the license.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.

continues on next page

Table 464 – continued from previous page

source	verb	target	doc
`ps:contact`	`-(has)>`	*	The contact is or was in possession of the target node.
`ps:contact`	`-(owns)>`	*	The contact owns or owned the target node.
`ps:person`	`-(has)>`	*	The person is or was in possession of the target node.
`ps:person`	`-(owns)>`	*	The person owns or owned the target node.
`risk:attack`	`-(targets)>`	*	The attack targeted the target node.
`risk:attack`	`-(uses)>`	*	The attack used the target node to facilitate the attack.
`risk:compromise`	`-(stole)>`	*	The target node was stolen or copied as a result of the compromise
`risk:extortion`	`-(leveraged)>`	*	The extortion event was based on attacker access to the target node
`risk:leak`	`-(leaked)>`	*	The leak included the disclosure of the target node.
`risk:threat`	`-(targets)>`	*	The threat cluster targeted the target node.
`risk:threat`	`-(uses)>`	*	The threat cluster uses the target node.
`risk:tool:software`	`-(uses)>`	*	The tool uses the target node.
`sci:evidence`	`-(has)>`	*	The evidence includes observations from the target nodes.
`sci:experiment`	`-(uses)>`	*	The experiment used the target nodes when it was run.
`sci:observation`	`-(has)>`	*	The observations are summarized from the target nodes.

transport:land:registration

Registration issued to a contact for a land vehicle.

The base type for the form can be found at *transport:land:registration*.

Properties:

name	type	doc
`:id`	*str* strip: `True`	The vehicle registration ID or license plate.
`:contact`	*ps:contact*	The contact info of the registrant.
`:license`	*transport:land:license*	The license used to register the vehicle.
`:issued`	*time*	The time the vehicle registration was issued.
`:expires`	*time*	The time the vehicle registration expires.
`:vehicle`	*transport:land:vehicle*	The vehicle being registered.
`:issuer`	*ou:org*	The org which issued the registration.
`:issuer:name`	*ou:name*	The name of the org which issued the registration.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:land:vehicle

An individual vehicle.

The base type for the form can be found at *transport:land:vehicle*.

Properties:

name	type	doc
:serial	*str* strip: True	The serial number or VIN of the vehicle.
:built	*time*	The date the vehicle was constructed.
:make	*ou:name*	The make of the vehicle.
:model	*str* lower: True onespace: True	The model of the vehicle.
:registration	*transport:land:registration*	The current vehicle registration information.
:owner	*ps:contact*	The contact info of the owner of the vehicle.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.

continues on next page

Table 466 – continued from previous page

source	verb	target	doc
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:sea:telem

A telemetry sample from a vessel in transit.

The base type for the form can be found at *transport:sea:telem*.

Properties:

name	type	doc
:vessel	*transport:sea:vessel*	The vessel being measured.
:time	*time*	The time the telemetry was sampled.
:latlong	*geo:latlong*	The lat/lon of the vessel at the time.
:loc	*loc*	The location of the vessel at the time.
:place	*geo:place*	The place that the lat/lon geocodes to.
:accuracy	*geo:dist*	The horizontal accuracy of the latlong sample.
:course	*transport:direction*	The direction, in degrees from true North, that the vessel is traveling.
:heading	*transport:direction*	The direction, in degrees from true North, that the bow of the vessel is pointed.
:speed	*velocity*	The speed of the vessel at the time.
:draft	*geo:dist*	The keel depth at the time.
:airdraft	*geo:dist*	The maximum height of the ship from the waterline.
:destination	*geo:place*	The fully resolved destination that the vessel has declared.
:destination:n	*geo:name*	The name of the destination that the vessel has declared.
:destination:e	*time*	The estimated time of arrival that the vessel has declared.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

transport:sea:vessel

An individual sea vessel.

The base type for the form can be found at *transport:sea:vessel*.

Properties:

name	type	doc
:imo	*transport:sea:imo*	The International Maritime Organization number for the vessel.
:name	*str* lower: True onespace: True	The name of the vessel.
:length	*geo:dist*	The official overall vessel length.
:beam	*geo:dist*	The official overall vessel beam.
:flag	*iso:3166:cc*	The country the vessel is flagged to.
:mmsi	*transport:sea:mmsi*	The Maritime Mobile Service Identifier assigned to the vessel.
:built	*time*	The year the vessel was constructed.
:make	*str* lower: True strip: True	The make of the vessel.
:model	*str* lower: True strip: True	The model of the vessel.
:operator	*ps:contact*	The contact information of the operator.

Source Edges:

source	verb	target	doc
*	-(meets)>	ou:requiremen	The requirement is met by the source node.
*	-(refs)>	*	The source node contains a reference to the target node.
*	-(seenat)>	geo:telem	The source node was seen at the geo:telem node place and time.

Target Edges:

source	verb	target	doc
*	-(refs)>	*	None
econ:purchase	-(acquired)>	*	The purchase was used to acquire the target node.
it:app:snort:rule	-(detects)>	*	The snort rule is intended for use in detecting the target node.
it:app:yara:rule	-(detects)>	*	The YARA rule is intended for use in detecting the target node.
it:exec:query	-(found)>	*	The target node was returned as a result of running the query.
meta:note	-(about)>	*	The meta:note is about the target node.
meta:rule	-(detects)>	*	The meta:rule is designed to detect instances of the target node.
meta:rule	-(matches)>	*	The meta:rule has matched on target node.
meta:source	-(seen)>	*	The meta:source observed the target node.
ou:campaign	-(targets)>	*	The campaign targeted the target nodes.
ou:campaign	-(uses)>	*	The campaign made use of the target node.
ou:contribution	-(includes)>	*	The contribution includes the specific node.
ou:org	-(has)>	*	The organization is or was in possession of the target node.
ou:org	-(owns)>	*	The organization owns or owned the target node.
ou:org	-(targets)>	*	The organization targets the target node.
ou:org	-(uses)>	*	The ou:org makes use of the target node.
ps:contact	-(has)>	*	The contact is or was in possession of the target node.
ps:contact	-(owns)>	*	The contact owns or owned the target node.
ps:person	-(has)>	*	The person is or was in possession of the target node.
ps:person	-(owns)>	*	The person owns or owned the target node.
risk:attack	-(targets)>	*	The attack targeted the target node.
risk:attack	-(uses)>	*	The attack used the target node to facilitate the attack.
risk:compromise	-(stole)>	*	The target node was stolen or copied as a result of the compromise
risk:extortion	-(leveraged)>	*	The extortion event was based on attacker access to the target node
risk:leak	-(leaked)>	*	The leak included the disclosure of the target node.
risk:threat	-(targets)>	*	The threat cluster targeted the target node.
risk:threat	-(uses)>	*	The threat cluster uses the target node.
risk:tool:software	-(uses)>	*	The tool uses the target node.
sci:evidence	-(has)>	*	The evidence includes observations from the target nodes.
sci:experiment	-(uses)>	*	The experiment used the target nodes when it was run.
sci:observation	-(has)>	*	The observations are summarized from the target nodes.

12.2.2 Universal Properties

Universal props are system level properties which may be present on every node.

These properties are not specific to a particular form and exist outside of a particular namespace.

.created

The time the node was created in the cortex. It has the following property options set:

- Read Only: `True`

The universal property type is *time*. Its type has the following options set:

- ismin: `True`

.seen

The time interval for first/last observation of the node.

The universal property type is *ival*.

12.3 Datamodel Deprecation Policy

As the Synapse Data Model has grown and evolved over time, Vertex has found the need to deprecate model elements which are no longer useful. These elements may represent relationships which are better captured with newer elements; concepts which are better represented by convention; or other issues. As such, model elements (types, forms, and properties) which are deprecated should no longer be used for new data modeling. Deprecated model elements will be removed in a future Synapse release, no earlier than `v3.0.0`.

For deprecated model elements, suggested alternatives will be provided and example Storm queries which can be used to migrate data in such a fashion.

12.3.1 Using Deprecated Model Elements

When Deprecated model elements are used in a Cortex, the following log events will be made:

- One startup, if a extended property definition uses a deprecated type to define it, a warning message will be logged.
- If a extended property is added which uses a deprecated type to define it, a warning message will be logged.
- Any Types or Forms, from a datamodel loaded by a custom CoreModule, which use a deprecated model component will cause a warning message to be logged. This includes any Array or Comp type model elements which utilize a deprecated Type.
- If a property or tag property is set on a node which is deprecated or using a deprecated type, that will cause a warning message to be logged and a `warn` message to be sent over the Storm runtime. This only occurs once per given runtime.
- If a node is made using deprecated form or using a deprecated type, that will cause a warning message to be logged and a `warn` message to be sent over the Storm runtime. This only occurs once per given runtime.

Deleting nodes which use deprecated model elements does not trigger warnings, since that would normally be done after an associated data migration and would be excessive in the event of a large migration.

12.3.2 Deprecated Model Elements

The following elements are deprecated.

Types

- ***file:string***
  - -(refs)> it:dev:str
- ***it:reveng:funcstr***
  - Please use the *:strings* array property on the *it:reveng:function* form.
- ***lang:idiom***

  - Please use *lang:translation* instead.
- ***lang:trans***
  - Please use *lang:translation* instead.
- ***ou:hasalias***
  - *ou:hasalias* is deprecated in favor of the *:alias* property on *ou:org* nodes.
- ***ou:meet:attendee***
  - *ou:meet:attendee* has been superseded by *ou:attendee*. *ou:attendee* has the *:meet* property to denote what meeting the attendee attended.
- ***ou:conference:attendee***
  - *ou:conference:attendee* has been superseded by *ou:attendee*. *ou:attendee* has the *:conference* property to denote what conference the attendee attended.
- ***ou:conference:event:attendee***
  - *ou:conference:attendee* has been superseded by *ou:attendee*. *ou:attendee* has the *:conference* property to denote what conference event the attendee attended.
- ***ou:member***
  - *ou:member* has been superseded by *ou:position*.
- ***ps:persona***
  - Please use the *ps:person* or *ps:contact* types.
- ***ps:person:has***
  - Please use *edge:has* or a light edge.
- ***ps:persona:has***
  - Please use *ps:person* or *ps:context* in combination with an *edge:has* or a light edge.
- ***inet:ssl:cert***
  - *inet:ssl:cert* is deprecated in favor of *inet:tls:servercert* and *inet:tls:clientcert*.

Forms

Consistent with the deprecated types, the following forms are deprecated: - *file:string* - *it:reveng:funcstr* - *lang:idiom* - *lang:trans* - *ou:hasalias* - *ou:meet:attendee* - *ou:conference:attendee* - *ou:conference:event:attendee* - *ou:member* - *ps:person:has* - *ps:persona* - *ps:persona:has* - *inet:ssl:cert*

Properties

- ***ps:person***
  - ***:img***
    - *ps:person:img* has been renamed to *ps:person:photo*.
- ***it:prod:soft***
  - ***author:org*, *author:acct*, *author:email*, and *author:person***
    - These properties have been collected into the *it:prod:soft:author* property, which is typed as a *ps:contact*.

- ***media:news***
  - ***:author***
    - The *media:news:author* property has been superseded by the array property of *media:news:authors*, which is an array of type *ps:contact*.
- ***file:subfile***
  - ***:name***
    - The *file:subfile:name* property has been superseded by the property *file:subfile:path*, which is typed as *file:path*.
- ***ou:org***
  - ***:naics* and *:sic***
    - The *ou:org:naics* and *ou:org:sic* properties has been collected into the *ou:org:industries* property, which is an array of type *ou:industry*.
  - ***:has***
    - Please use an *edge:has* node or a light edge.
- ***risk:attack***
  - ***:actor:org***
    - Please use the *:attacker ps:contact* property to allow entity resolution.
  - ***:actor:person***
    - Please use the *:attacker ps:contact* property to allow entity resolution.
  - ***:target:org***
    - Please use the *:target ps:contact* property to allow entity resolution.
  - ***:target:person***
    - Please use the *:target ps:contact* property to allow entity resolution.
- ***ou:campaign***
  - ***:type***
    - Please use the *:camptype taxonomy* property.
- ***it:host***
  - ***:manu***
    - This property has been superseded by the *it:prod:hardware:make* property, which is typed as *ou:name*.
  - ***:model***
    - This property has been superseded by the *it:prod:hardware:model* property, which is typed as string.
- ***it:exec:proc***
  - ***:user***
    - Please use the *:account it:exec:proc* property to link processes to users.

CHAPTER

THIRTEEN

STORM LIBRARY DOCUMENTATION

This contains API documentation for Storm Libraries and Storm Types.

Storm Types (also called Storm Objects) are objects in the Storm Runtime that can represent values such as nodes in the runtime or objects in the Cortex. Storm Types encompass objects from strings of characters (*str*), to objects representing Cron Jobs in the Cortex (*cronjob*), to nodes in the Cortex (*node*). These objects each have their own properties and methods defined on them that can be used to inspect or edit that object. For instance, String Storm Types all have the `upper()` method defined on them that returns a new instance of that String, except with every letter turned uppercase (*upper()*). Storm Types help form the basis for programmatic manipulation of objects and data in the Cortex.

Storm Libraries are ready-made tools in the Storm query language for creating, updating, or fetching data using Storm Types. Storm libraries include functionality for making HTTP requests (via *$lib.inet.http*), scraping nodes from text (*$lib.scrape*), manipulating Cortex objects such as Queues (*$lib.queue*) and StormDmons (*$lib.dmon*), creating new Cron Jobs (*$lib.cron*), and more. Many of these libraries accept or return Storm Types as part of their usage. For instance, there is a library in Storm for interacting with OAuthv1 servers (*$lib.inet.http.oauth.v1.client(ckey, csecret, atoken, asecret, sigtype=QUERY)*), and it accepts several String Storm Types as parameters and returns an OAuthV1 client object for later usage (*inet:http:oauth:v1:client*).

Storm Libraries form a powerful bench of tools for usage within the Storm query language.

The current sections are:

13.1 Storm Libraries

Storm Libraries represent powerful tools available inside of the Storm query language.

13.1.1 $lib

The Base Storm Library. This mainly contains utility functionality.

$lib.cast(name, valu)

Normalize a value as a Synapse Data Model Type.

Args:
name (str): The name of the model type to normalize the value as.

valu (any): The value to normalize.

Returns:
The normalized value. The return type is `prim`.

$lib.copy(item)

Create and return a deep copy of the given storm object.

Note:
This is currently limited to msgpack compatible primitives.

Examples:
Make a copy of a list or dict:

```
$copy = $lib.copy($item)
```

Args:
item (prim): The item to make a copy of.

Returns:
A deep copy of the primitive object. The return type is `prim`.

$lib.debug

True if the current runtime has debugging enabled.

Note:
The debug state is inherited by sub-runtimes at instantiation time. Any changes to a runtime's debug state do not percolate automatically.

Examples:
Check if the runtime is in debug and print a message:

```
if $lib.debug {
    $lib.print('Doing stuff!")
}
```

Update the current runtime to enable debugging:

```
$lib.debug = $lib.true
```

Returns:
The return type is *boolean*. When this is used to set the value, it does not have a return type.

$lib.exit(mesg=$lib.null, **kwargs)

Cause a Storm Runtime to stop running.

Args:
mesg (str): Optional string to warn.

**kwargs (any): Keyword arguments to substitute into the mesg.

Returns:
The return type is `null`.

$lib.false

This constant represents a value of False that can be used in Storm.

Examples:
Conditionally print a statement based on the constant value:

```
cli> storm if $lib.false { $lib.print('Is True') } else { $lib.print('Is False') }
Is False
```

Returns:
The type is *boolean*.

$lib.fire(name, **info)

Fire an event onto the runtime.

Notes:
This fires events as `storm:fire` event types. The name of the event is placed into a `type` key, and any additional keyword arguments are added to a dictionary under the `data` key.

Examples:
Fire an event called `demo` with some data:

```
cli> storm $foo='bar' $lib.fire('demo', foo=$foo, knight='ni')
...
('storm:fire', {'type': 'demo', 'data': {'foo': 'bar', 'knight': 'ni'}})
...
```

Args:
name (str): The name of the event to fire.

**info (any): Additional keyword arguments containing data to add to the event.

Returns:
The return type is `null`.

$lib.guid(*args, valu=$lib.undef)

Get a random guid, or generate a guid from the arguments.

Args:

*args (prim): Arguments which are hashed to create a guid.

valu (prim): Create a guid from a single value (no positional arguments can be specified).

Returns:

A guid. The return type is *str*.

$lib.import(name, debug=$lib.false, reqvers=$lib.null)

Import a Storm module.

Args:

name (str): Name of the module to import.

debug (boolean): Enable debugging in the module.

reqvers (str): Version requirement for the imported module.

Returns:

A `lib` instance representing the imported package. The return type is `lib`.

$lib.len(item)

Get the length of a item.

This could represent the size of a string, or the number of keys in a dictionary, or the number of elements in an array. It may also be used to iterate an emitter or yield function and count the total.

Args:

item (prim): The item to get the length of.

Returns:

The length of the item. The return type is `int`.

$lib.list(*vals)

Get a Storm List object.

Args:

*vals (any): Initial values to place in the list.

Returns:

A new list object. The return type is *list*.

$lib.max(*args)

Get the maximum value in a list of arguments.

Args:
*args (any): List of arguments to evaluate.

Returns:
The largest argument. The return type is `int`.

$lib.min(*args)

Get the minimum value in a list of arguments.

Args:
*args (any): List of arguments to evaluate.

Returns:
The smallest argument. The return type is `int`.

$lib.null

This constant represents a value of None that can be used in Storm.

Examples:
Create a dictionary object with a key whose value is null, and call `$lib.fire()` with it:

```
cli> storm $d=({"key": $lib.null}) $lib.fire('demo', d=$d)
('storm:fire', {'type': 'demo', 'data': {'d': {'key': None}}})
```

Returns:
The type is `null`.

$lib.pprint(item, prefix=, clamp=$lib.null)

The pprint API should not be considered a stable interface.

Args:
item (any): Item to pprint

prefix (str): Line prefix.

clamp (int): Line clamping length.

Returns:
The return type is `null`.

$lib.print(mesg, **kwargs)

Print a message to the runtime.

Examples:

Print a simple string:

```
cli> storm $lib.print("Hello world!")
Hello world!
```

Format and print string based on variables:

```
cli> storm $d=({"key1": (1), "key2": "two"})
     for ($key, $value) in $d { $lib.print('{k} => {v}', k=$key, v=$value) }
key1 => 1
key2 => two
```

Use values off of a node to format and print string:

```
cli> storm inet:ipv4:asn
     $lib.print("node: {ndef}, asn: {asn}", ndef=$node.ndef(), asn=:asn) | spin
node: ('inet:ipv4', 16909060), asn: 1138
```

Notes:

Arbitrary objects can be printed as well. They will have their Python __repr()__ printed.

Args:

mesg (str): String to print.

**kwargs (any): Keyword arguments to substitute into the mesg.

Returns:

The return type is `null`.

$lib.raise(name, mesg, **info)

Raise an exception in the storm runtime.

Args:

name (str): The name of the error condition to raise.

mesg (str): A friendly description of the specific error.

**info (any): Additional metadata to include in the exception.

Returns:

This function does not return. The return type is `null`.

$lib.range(stop, start=$lib.null, step=$lib.null)

Generate a range of integers.

Examples:
Generate a sequence of integers based on the size of an array:

```
cli> storm $a=(foo,bar,(2)) for $i in $lib.range($lib.len($a)) {$lib.fire('test',␣
↪indx=$i, valu=$a.$i)}
Executing query at 2021/03/22 19:25:48.835
('storm:fire', {'type': 'test', 'data': {'index': 0, 'valu': 'foo'}})
('storm:fire', {'type': 'test', 'data': {'index': 1, 'valu': 'bar'}})
('storm:fire', {'type': 'test', 'data': {'index': 2, 'valu': 2}})
```

Notes:
The range behavior is the same as the Python3 `range()` builtin Sequence type.

Args:
stop (int): The value to stop at.

start (int): The value to start at.

step (int): The range step size.

Yields:
The sequence of integers. The return type is `int`.

$lib.set(*vals)

Get a Storm Set object.

Args:
*vals (any): Initial values to place in the set.

Returns:
The new set. The return type is *set*.

$lib.sorted(valu, reverse=$lib.false)

Yield sorted values.

Args:
valu (any): An iterable object to sort.

reverse (boolean): Reverse the sort order.

Yields:
Yields the sorted output. The return type is `any`.

$lib.text(*args)

Get a Storm Text object.

Args:
*args (str): An initial set of values to place in the Text. These values are joined together with an empty string.

Returns:
The new Text object. The return type is *text*.

$lib.true

This constant represents a value of True that can be used in Storm.

Examples:
Conditionally print a statement based on the constant value:

```
cli> storm if $lib.true { $lib.print('Is True') } else { $lib.print('Is False') }
Is True
```

Returns:
The type is *boolean*.

$lib.trycast(name, valu)

Attempt to normalize a value and return status and the normalized value.

Examples:
Do something if the value is a valid IPV4:

```
($ok, $ipv4) = $lib.trycast(inet:ipv4, 1.2.3.4)
if $ok { $dostuff($ipv4) }
```

Args:
name (str): The name of the model type to normalize the value as.

valu (any): The value to normalize.

Returns:
A list of (<bool>, <prim>) for status and normalized value. The return type is *list*.

$lib.undef

This constant can be used to unset variables and derefs.

Examples:
Unset the variable $foo:

```
$foo = $lib.undef
```

Remove a dictionary key bar:

```
$foo.bar = $lib.undef
```

Remove a list index of 0:

```
$foo.0 = $lib.undef
```

Returns:
The type is `undef`.

$lib.warn(mesg, **kwargs)

Print a warning message to the runtime.

Notes:
Arbitrary objects can be warned as well. They will have their Python __repr()__ printed.

Args:
mesg (str): String to warn.

**kwargs (any): Keyword arguments to substitute into the mesg.

Returns:
The return type is `null`.

13.1.2 $lib.aha

A Storm Library for interacting with AHA.

$lib.aha.del(svcname)

Delete a service from AHA.

Examples:
Deleting a service with its relative name:

```
$lib.aha.del(00.mysvc...)
```

Deleting a service with its full name:

```
$lib.aha.del(00.mysvc.loop.vertex.link)
```

Args:
svcname (str): The name of the service to delete. It is easiest to use the relative name of a service, ending with "...".

Returns:
The return type is `null`.

$lib.aha.get(svcname, filters=$lib.null)

Get information about an AHA service.

Examples:
Getting service information with a relative name:

```
$lib.aha.get(00.cortex...)
```

Getting service information with its full name:

```
$lib.aha.get(00.cortex.loop.vertex.link)
```

Args:
svcname (str): The name of the AHA service to look up. It is easiest to use the relative name of a service, ending with "…".

filters (dict): An optional dictionary of filters to use when resolving the AHA service.

Returns:
The AHA service information dictionary, or $lib.null. The return type may be one of the following: `null`, *dict*.

$lib.aha.list()

Enumerate all of the AHA services.

Yields:
The AHA service dictionaries. The return type is *list*.

13.1.3 $lib.aha.pool

A Storm Library for interacting with AHA service pools.

$lib.aha.pool.add(name)

Add a new AHA service pool.

Examples:
Add a pool via its relative name:

```
$lib.aha.pool.add(pool00.cortex...)
```

Args:
name (str): The name of the pool to add. It is easiest to use the relative name of a pool, ending with "…".

Returns:
The return type is *aha:pool*.

$lib.aha.pool.del(name)

Delete an existing AHA service pool.

Examples:
Delete a pool via its relative name:

```
$lib.aha.pool.del(pool00.cortex...)
```

Args:
name (str): The name of the pool to delete. It is easiest to use the relative name of a pool, ending with "…".

Returns:
The AHA pool definition that was deleted. The return type is *dict*.

$lib.aha.pool.get(name)

Get an existing AHA service pool.

Args:
name (str): The name of the pool to get. It is easiest to use the relative name of a pool, ending with "…".

Returns:
The pool if it exists, or $lib.null. The return type may be one of the following: `null`, *aha:pool*.

$lib.aha.pool.list()

Enumerate all of the AHA service pools.

Yields:
The return type is *aha:pool*.

13.1.4 $lib.auth

A Storm Library for interacting with Auth in the Cortex.

$lib.auth.getPermDef(perm)

Return a single permission definition.

Args:
perm (list): A permission tuple.

Returns:
A permission definition or null. The return type is *dict*.

$lib.auth.getPermDefs()

Return a list of permission definitions.

Returns:
The list of permission definitions. The return type is *list*.

$lib.auth.ruleFromText(text)

Get a rule tuple from a text string.

Args:
text (str): The string to process.

Returns:
A tuple containing a bool and a list of permission parts. The return type is *list*.

$lib.auth.textFromRule(rule)

Return a text string from a rule tuple.

Args:
rule (list): A rule tuple.

Returns:
The rule text. The return type is *str*.

13.1.5 $lib.auth.easyperm

A Storm Library for interacting with easy perm dictionaries.

$lib.auth.easyperm.allowed(edef, level)

Check if the current user has a permission level in an easy perm dictionary.

Args:
edef (dict): The easy perm dictionary to check.

level (str): The required permission level number.

Returns:
True if the user meets the requirement, false otherwise. The return type is *boolean*.

$lib.auth.easyperm.confirm(edef, level, mesg=$lib.null)

Require that the current user has a permission level in an easy perm dictionary.

Args:
edef (dict): The easy perm dictionary to check.

level (str): The required permission level number.

mesg (str): Optional error message to present if user does not have required permission level.

Returns:
The return type is `null`.

$lib.auth.easyperm.init(edef=$lib.null, default=(1))

Add the easy perm structure to a new or existing dictionary.

Note:
The current user will be given admin permission in the new easy perm structure.

Args:
edef (dict): A dictionary to add easy perms to.

default (int): Specify the default permission level for this item.

Returns:
Dictionary with the easy perm structure. The return type is *dict*.

$lib.auth.easyperm.set(edef, scope, iden, level)

Set the permission level for a user or role in an easy perm dictionary.

Args:
edef (dict): The easy perm dictionary to modify.

scope (str): The scope, either "users" or "roles".

iden (str): The user/role iden depending on scope.

level (int): The permission level number, or None to remove the permission.

Returns:
Dictionary with the updated easy perm structure. The return type is *dict*.

13.1.6 $lib.auth.gates

A Storm Library for interacting with Auth Gates in the Cortex.

$lib.auth.gates.get(iden)

Get a specific Gate by iden.

Args:
iden (str): The iden of the gate to retrieve.

Returns:
The `auth:gate` if it exists, otherwise null. The return type may be one of the following: `null`, *auth:gate*.

$lib.auth.gates.list()

Get a list of Gates in the Cortex.

Returns:
A list of `auth:gate` objects. The return type is *list*.

13.1.7 $lib.auth.roles

A Storm Library for interacting with Auth Roles in the Cortex.

$lib.auth.roles.add(name)

Add a Role to the Cortex.

Args:
name (str): The name of the role.

Returns:
The new role object. The return type is *auth:role*.

$lib.auth.roles.byname(name)

Get a specific Role by name.

Args:
name (str): The name of the role to retrieve.

Returns:
The role by name, or null if it does not exist. The return type may be one of the following: `null`, *auth:role*.

$lib.auth.roles.del(iden)

Delete a Role from the Cortex.

Args:
iden (str): The iden of the role to delete.

Returns:
The return type is `null`.

$lib.auth.roles.get(iden)

Get a specific Role by iden.

Args:
iden (str): The iden of the role to retrieve.

Returns:
The `auth:role` object; or null if the role does not exist. The return type may be one of the following: `null`, *auth:role*.

$lib.auth.roles.list()

Get a list of Roles in the Cortex.

Returns:
A list of `auth:role` objects. The return type is *list*.

13.1.8 $lib.auth.users

A Storm Library for interacting with Auth Users in the Cortex.

$lib.auth.users.add(name, passwd=$lib.null, email=$lib.null, iden=$lib.null)

Add a User to the Cortex.

Args:
name (str): The name of the user.

passwd (str): The user's password.

email (str): The user's email address.

iden (str): The iden to use to create the user.

Returns:
The `auth:user` object for the new user. The return type is *auth:user*.

$lib.auth.users.byname(name)

Get a specific user by name.

Args:
name (str): The name of the user to retrieve.

Returns:
The `auth:user` object, or none if the user does not exist. The return type may be one of the following: `null`, *auth:user*.

$lib.auth.users.del(iden)

Delete a User from the Cortex.

Args:
iden (str): The iden of the user to delete.

Returns:
The return type is `null`.

$lib.auth.users.get(iden)

Get a specific User by iden.

Args:
iden (str): The iden of the user to retrieve.

Returns:
The `auth:user` object, or none if the user does not exist. The return type may be one of the following: `null`, *auth:user*.

$lib.auth.users.list()

Get a list of Users in the Cortex.

Returns:
A list of `auth:user` objects. The return type is *list*.

13.1.9 $lib.axon

A Storm library for interacting with the Cortex's Axon.

For APIs that accept an ssl_opts argument, the dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can be␣
↪included in client_cert.
}
```

$lib.axon.csvrows(sha256, dialect=excel, errors=ignore, **fmtparams)

Yields CSV rows from a CSV file stored in the Axon.

Notes:
The dialect and fmtparams expose the Python csv.reader() parameters.

Example:
Get the rows from a given csv file:

```
for $row in $lib.axon.csvrows($sha256) {
    $dostuff($row)
}
```

Get the rows from a given tab separated file:

```
for $row in $lib.axon.csvrows($sha256, delimiter="\t") {
    $dostuff($row)
}
```

Args:
sha256 (str): The SHA256 hash of the file.

dialect (str): The default CSV dialect to use.

errors (str): Specify how encoding errors should handled.

**fmtparams (any): Format arguments.

Yields:
A list of strings from the CSV file. The return type is *list*.

$lib.axon.del(sha256)

Remove the bytes from the Cortex's Axon by sha256.

Example:
Delete files from the axon based on a tag:

```
file:bytes#foo +:sha256 $lib.axon.del(:sha256)
```

Args:
sha256 (hash:sha256): The sha256 of the bytes to remove from the Axon.

Returns:
True if the bytes were found and removed. The return type is *boolean*.

$lib.axon.dels(sha256s)

Remove multiple byte blobs from the Cortex's Axon by a list of sha256 hashes.

Example:
Delete a list of files (by hash) from the Axon:

```
$list = ($hash0, $hash1, $hash2)
$lib.axon.dels($list)
```

Args:
sha256s (list): A list of sha256 hashes to remove from the Axon.

Returns:
A list of boolean values that are True if the bytes were found. The return type is *list*.

$lib.axon.has(sha256)

Check if the Axon the Cortex is configured to use has a given sha256 value.

Examples:
Check if the Axon has a given file:

```
# This example assumes the Axon does have the bytes
cli> storm if $lib.axon.
↪has(9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08) {
        $lib.print("Has bytes")
    } else {
        $lib.print("Does not have bytes")
    }

Has bytes
```

Args:
sha256 (str): The sha256 value to check.

Returns:
True if the Axon has the file, false if it does not. The return type is *boolean*.

$lib.axon.hashset(sha256)

Return additional hashes of the bytes stored in the Axon for the given sha256.

Examples:
Get the md5 hash for a file given a variable named `$sha256`:

```
$hashset = $lib.axon.hashset($sha256)
$md5 = $hashset.md5
```

Args:
sha256 (str): The sha256 value to calculate hashes for.

Returns:
A dictionary of additional hashes. The return type is *dict*.

$lib.axon.jsonlines(sha256, errors=ignore)

Yields JSON objects from a JSON-lines file stored in the Axon.

Example:
Get the JSON objects from a given JSONL file:

```
for $item in $lib.axon.jsonlines($sha256) {
    $dostuff($item)
}
```

Args:
sha256 (str): The SHA256 hash of the file.

errors (str): Specify how encoding errors should handled.

Yields:
A JSON object parsed from a line of text. The return type is any.

$lib.axon.list(offs=(0), wait=$lib.false, timeout=$lib.null)

List (offset, sha256, size) tuples for files in the Axon in added order.

Example:
List files:

```
for ($offs, $sha256, $size) in $lib.axon.list() {
    $lib.print($sha256)
}
```

Start list from offset 10:

```
for ($offs, $sha256, $size) in $lib.axon.list(10) {
    $lib.print($sha256)
}
```

Args:
offs (int): The offset to start from.

wait (boolean): Wait for new results and yield them in realtime.

timeout (int): The maximum time to wait for a new result before returning.

Yields:
Tuple of (offset, sha256, size) in added order. The return type is *list*.

$lib.axon.metrics()

Get runtime metrics of the Axon.

Example:
Print the total number of files stored in the Axon:

```
$data = $lib.axon.metrics()
$lib.print("The Axon has {n} files", n=$data."file:count")
```

Returns:
A dictionary containing runtime data about the Axon. The return type is *dict*.

$lib.axon.put(byts)

Save the given bytes variable to the Axon the Cortex is configured to use.

Examples:
Save a base64 encoded buffer to the Axon:

```
cli> storm $s='dGVzdA==' $buf=$lib.base64.decode($s) ($size, $sha256)=$lib.axon.put(
↪$buf)
     $lib.print('size={size} sha256={sha256}', size=$size, sha256=$sha256)

size=4 sha256=9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
```

Args:
byts (bytes): The bytes to save.

Returns:
A tuple of the file size and sha256 value. The return type is *list*.

$lib.axon.readlines(sha256, errors=ignore)

Yields lines of text from a plain-text file stored in the Axon.

Examples:

// Get the lines for a given file. for $line in $lib.axon.readlines($sha256) {

$dostuff($line)

}

Args:
sha256 (str): The SHA256 hash of the file.

errors (str): Specify how encoding errors should handled.

Yields:
A line of text from the file. The return type is *str*.

$lib.axon.size(sha256)

Return the size of the bytes stored in the Axon for the given sha256.

Examples:
Get the size for a file given a variable named `$sha256`:

```
$size = $lib.axon.size($sha256)
```

Args:
sha256 (str): The sha256 value to check.

Returns:
The size of the file or `null` if the file is not found. The return type may be one of the following: `int`, `null`.

$lib.axon.upload(genr)

Upload a stream of bytes to the Axon as a file.

Examples:
Upload bytes from a generator:

```
($size, $sha256) = $lib.axon.upload($getBytesChunks())
```

Args:
genr (generator): A generator which yields bytes.

Returns:
A tuple of the file size and sha256 value. The return type is *list*.

$lib.axon.urlfile(*args, **kwargs)

Retrieve the target URL using the wget() function and construct an inet:urlfile node from the response.

Notes:
This accepts the same arguments as `$lib.axon.wget()`.

Args:
*args (any): Args from `$lib.axon.wget()`.

**kwargs (any): Args from `$lib.axon.wget()`.

Returns:
The `inet:urlfile` node on success, `null` on error. The return type may be one of the following: *node*, `null`.

$lib.axon.wget(url, headers=$lib.null, params=$lib.null, method=GET, json=$lib.null, body=$lib.null, ssl=$lib.true, timeout=$lib.null, proxy=$lib.null, ssl_opts=$lib.null)

A method to download an HTTP(S) resource into the Cortex's Axon.

Notes:
The response body will be stored regardless of the status code. See the `Axon.wget()` API documentation to see the complete structure of the response dictionary.

Example:
Get the Vertex Project website:

```
$headers = ({})
$headers."User-Agent" = Foo/Bar

$resp = $lib.axon.wget("http://vertex.link", method=GET, headers=$headers)
if $resp.ok { $lib.print("Downloaded: {size} bytes", size=$resp.size) }
```

Args:
url (str): The URL to download

headers (dict): An optional dictionary of HTTP headers to send.

params (dict): An optional dictionary of URL parameters to add.

method (str): The HTTP method to use.

json (dict): A JSON object to send as the body.

body (bytes): Bytes to send as the body.

ssl (boolean): Set to False to disable SSL/TLS certificate verification.

timeout (int): Timeout for the download operation.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.axon help for additional details.

Returns:
A status dictionary of metadata. The return type is *dict*.

$lib.axon.wput(sha256, url, headers=$lib.null, params=$lib.null, method=PUT, ssl=$lib.true, timeout=$lib.null, proxy=$lib.null, ssl_opts=$lib.null)

A method to upload a blob from the axon to an HTTP(S) endpoint.

Args:
sha256 (str): The sha256 of the file blob to upload.

url (str): The URL to upload the file to.

headers (dict): An optional dictionary of HTTP headers to send.

params (dict): An optional dictionary of URL parameters to add.

method (str): The HTTP method to use.

ssl (boolean): Set to False to disable SSL/TLS certificate verification.

timeout (int): Timeout for the download operation.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.axon help for additional details.

Returns:
A status dictionary of metadata. The return type is *dict*.

13.1.10 $lib.backup

A Storm Library for interacting with the backup APIs in the Cortex.

$lib.backup.del(name)

Remove a backup by name.

Args:
name (str): The name of the backup to remove.

Returns:
The return type is `null`.

$lib.backup.list()

Get a list of backup names.

Returns:

A list of backup names. The return type is *list*.

$lib.backup.run(name=$lib.null, wait=$lib.true)

Run a Cortex backup.

Args:

name (str): The name of the backup to generate.

wait (boolean): If true, wait for the backup to complete before returning.

Returns:

The name of the newly created backup. The return type is *str*.

13.1.11 $lib.base64

A Storm Library for encoding and decoding base64 data.

$lib.base64.decode(valu, urlsafe=$lib.true)

Decode a base64 string into a bytes object.

Args:

valu (str): The string to decode.

urlsafe (boolean): Perform the decoding in a urlsafe manner if true.

Returns:

A bytes object for the decoded data. The return type is *bytes*.

$lib.base64.encode(valu, urlsafe=$lib.true)

Encode a bytes object to a base64 encoded string.

Args:

valu (bytes): The object to encode.

urlsafe (boolean): Perform the encoding in a urlsafe manner if true.

Returns:

A base64 encoded string. The return type is *str*.

13.1.12 $lib.basex

A Storm library which implements helpers for encoding and decoding strings using an arbitrary charset.

$lib.basex.decode(text, charset)

Decode a baseX string into bytes.

Args:

text (str): The hex string to be decoded into bytes.

charset (str): The charset used to decode the string.

Returns:

The decoded bytes. The return type is *bytes*.

$lib.basex.encode(byts, charset)

Encode bytes into a baseX string.

Args:

byts (bytes): The bytes to be encoded into a string.

charset (str): The charset used to encode the bytes.

Returns:

The encoded string. The return type is *str*.

13.1.13 $lib.bytes

A Storm Library for interacting with bytes storage. This Library is deprecated; use `$lib.axon.*` instead.

$lib.bytes.has(sha256)

Check if the Axon the Cortex is configured to use has a given sha256 value.

Examples:

Check if the Axon has a given file:

```
# This example assumes the Axon does have the bytes
cli> storm if $lib.bytes.
↪has(9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08) {
        $lib.print("Has bytes")
    } else {
        $lib.print("Does not have bytes")
    }

Has bytes
```

Args:

sha256 (str): The sha256 value to check.

Returns:

True if the Axon has the file, false if it does not. The return type is *boolean*.

$lib.bytes.hashset(sha256)

Return additional hashes of the bytes stored in the Axon for the given sha256.

Examples:
Get the md5 hash for a file given a variable named `$sha256`:

```
$hashset = $lib.bytes.hashset($sha256)
$md5 = $hashset.md5
```

Args:
sha256 (str): The sha256 value to calculate hashes for.

Returns:
A dictionary of additional hashes. The return type is *dict*.

$lib.bytes.put(byts)

Save the given bytes variable to the Axon the Cortex is configured to use.

Examples:
Save a base64 encoded buffer to the Axon:

```
cli> storm $s='dGVzdA==' $buf=$lib.base64.decode($s) ($size, $sha256)=$lib.bytes.
↪put($buf)
     $lib.print('size={size} sha256={sha256}', size=$size, sha256=$sha256)

size=4 sha256=9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
```

Args:
byts (bytes): The bytes to save.

Returns:
A tuple of the file size and sha256 value. The return type is *list*.

$lib.bytes.size(sha256)

Return the size of the bytes stored in the Axon for the given sha256.

Examples:
Get the size for a file given a variable named `$sha256`:

```
$size = $lib.bytes.size($sha256)
```

Args:
sha256 (str): The sha256 value to check.

Returns:
The size of the file or `null` if the file is not found. The return type may be one of the following: `int`, `null`.

$lib.bytes.upload(genr)

Upload a stream of bytes to the Axon as a file.

Examples:
Upload bytes from a generator:

```
($size, $sha256) = $lib.bytes.upload($getBytesChunks())
```

Args:
genr (generator): A generator which yields bytes.

Returns:
A tuple of the file size and sha256 value. The return type is *list*.

13.1.14 $lib.cache

A Storm Library for interacting with Cache Objects.

$lib.cache.fixed(callback, size=(10000))

Get a new Fixed Cache object.

On a cache-miss when calling .get(), the callback Storm query is executed in a sub-runtime in the current execution context. A special variable, $cache_key, will be set to the key argument provided to .get().

The callback Storm query must contain a return statement, and if it does not return a value when executed with the input, $lib.null will be set as the value.

The fixed cache uses FIFO to evict items once the maximum size is reached.

Examples:

// Use a callback query with a function that modifies the outer runtime, // since it will run in the scope where it was defined. $test = foo

function callback(key) {
$test = $key // this will modify $test in the outer runtime return(*{$key}-val*)

}

$cache = $lib.cache.fixed(${ return($callback($cache_key)) }) $value = $cache.get(bar) $lib.print($test) // this will equal "bar"

// Use a callback query that will not modify the outer runtime, // except for variables accessible as references. $test = foo $tests = ([])

$cache = $lib.cache.fixed(${
$test = $cache_key // this will *not* modify $test in the outer runtime $tests.append($cache_key) // this will modify $tests in the outer runtime return(*{$cache_key}-val*)

})

$value = $cache.get(bar) $lib.print($test) // this will equal "foo" $lib.print($tests) // this will equal (foo,)

Args:
callback: A Storm query that will return a value for $cache_key on a cache miss. The input type may be one of the following: `str`, `storm:query`.

size (int): The maximum size of the cache.

Returns:
A new `cache:fixed` object. The return type is *cache:fixed*.

13.1.15 $lib.cell

A Storm Library for interacting with the Cortex.

$lib.cell.getBackupInfo()

Get information about recent backup activity.

Returns:
A dictionary containing backup information. The return type is *dict*.

$lib.cell.getCellInfo()

Return metadata specific for the Cortex.

Returns:
A dictionary containing metadata. The return type is *dict*.

$lib.cell.getHealthCheck()

Get healthcheck information about the Cortex.

Returns:
A dictionary containing healthcheck information. The return type is *dict*.

$lib.cell.getMirrorUrls(name=$lib.null)

Get mirror Telepath URLs for an AHA configured service.

Args:
name (str): The name, or iden, of the service to get mirror URLs for (defaults to the Cortex if not provided).

Returns:
A list of Telepath URLs. The return type is *list*.

$lib.cell.getSystemInfo()

Get info about the system in which the Cortex is running.

Returns:
A dictionary containing system information. The return type is *dict*.

$lib.cell.hotFixesApply()

Apply known data migrations and fixes via storm.

Returns:
Tuple containing the current version after applying the fixes. The return type is *list*.

$lib.cell.hotFixesCheck()

Check to see if there are known hot fixes to apply.

Returns:
Bool indicating if there are hot fixes to apply or not. The return type is *boolean*.

$lib.cell.trimNexsLog(consumers=$lib.null, timeout=(30))

Rotate and cull the Nexus log (and any consumers) at the current offset.

If the consumers argument is provided they will first be checked if online before rotating and raise otherwise. After rotation, all consumers provided must catch-up to the offset to cull at within the specified timeout before executing the cull, and will raise otherwise.

Args:
consumers (array): List of Telepath URLs for consumers of the Nexus log.

timeout (int): Time (in seconds) to wait for consumers to catch-up before culling.

Returns:
The offset that was culled (up to and including). The return type is `int`.

$lib.cell.uptime(name=$lib.null)

Get update data for the Cortex or a connected Service.

Args:
name (str): The name, or iden, of the service to get uptime data for (defaults to the Cortex if not provided).

Returns:
A dictionary containing uptime data. The return type is *dict*.

13.1.16 $lib.compression.bzip2

A Storm library which implements helpers for bzip2 compression.

$lib.compression.bzip2.en(valu)

Compress bytes using bzip2 and return them.

Example:
Compress bytes with bzip2:

```
$foo = $lib.compression.bzip2.en($mybytez)
```

Args:
valu (bytes): The bytes to be compressed.

Returns:
The bzip2 compressed bytes. The return type is *bytes*.

$lib.compression.bzip2.un(valu)

Decompress bytes using bzip2 and return them.

Example:
Decompress bytes with bzip2:

```
$foo = $lib.compression.bzip2.un($mybytez)
```

Args:
valu (bytes): The bytes to be decompressed.

Returns:
Decompressed bytes. The return type is *bytes*.

13.1.17 $lib.compression.gzip

A Storm library which implements helpers for gzip compression.

$lib.compression.gzip.en(valu)

Compress bytes using gzip and return them.

Example:
Compress bytes with gzip:

```
$foo = $lib.compression.gzip.en($mybytez)
```

Args:
valu (bytes): The bytes to be compressed.

Returns:
The gzip compressed bytes. The return type is *bytes*.

$lib.compression.gzip.un(valu)

Decompress bytes using gzip and return them.

Example:
Decompress bytes with gzip:

```
$foo = $lib.compression.gzip.un($mybytez)
```

Args:
valu (bytes): The bytes to be decompressed.

Returns:
Decompressed bytes. The return type is *bytes*.

13.1.18 $lib.compression.zlib

A Storm library which implements helpers for zlib compression.

$lib.compression.zlib.en(valu)

Compress bytes using zlib and return them.

Example:
Compress bytes with zlib:

```
$foo = $lib.compression.zlib.en($mybytez)
```

Args:
valu (bytes): The bytes to be compressed.

Returns:
The zlib compressed bytes. The return type is *bytes*.

$lib.compression.zlib.un(valu)

Decompress bytes using zlib and return them.

Example:
Decompress bytes with zlib:

```
$foo = $lib.compression.zlib.un($mybytez)
```

Args:
valu (bytes): The bytes to be decompressed.

Returns:
Decompressed bytes. The return type is *bytes*.

13.1.19 $lib.cortex.httpapi

Library for interacting with the Extended HTTP API.

$lib.cortex.httpapi.add(path, name=, desc=, runas=owner, authenticated=$lib.true, readonly=$lib.false)

Add an Extended HTTP API endpoint to the Cortex.

This can be used to add an API endpoint which will be resolved under the API path "/api/ext/". New API endpoint objects are appended to a list of APIs to resolve in order.

Notes:
The Cortex does not make any attempt to do any inspection of path values which may conflict between one another. This is because the paths for a given endpoint may be changed, they can contain regular expressions, and they may have their resolution order changed. Cortex administrators are responsible for configuring their Extended HTTP API endpoints with correct paths and order to meet their use cases.

Example:
Add a simple API handler:

```
// Create an endpoint for /api/ext/foo/bar
$api = $lib.cortex.httpapi.add('foo/bar')

// Define a GET response handler via storm that makes a simple reply.
$api.methods.get = ${ $request.reply(200, body=({"some": "data}) }
```

Add a wildcard handler:

```
// Create a wildcard endpoint for /api/ext/some/thing([a-zA-Z0-9]*)/([a-zA-Z0-9]*)
$api = $lib.cortex.httpapi.add('some/thing([a-zA-Z0-9]*)/([a-zA-Z0-9]*)')

// The capture groups are exposed as request arguments.
// Echo them back to the caller.
$api.methods.get = ${
    $request.reply(200, body=({"args": $request.args})
}
```

Args:

path (string): The extended HTTP API path.

name (string): Friendly name for the Extended HTTP API

desc (string): Description for the Extended HTTP API.

runas (string): Run the storm query as the API “owner” or as the authenticated “user”.

authenticated (boolean): Require the API endpoint to be authenticated.

readonly (boolean): Run the Extended HTTP Storm methods in readonly mode.

Returns:

A new `http:api` object. The return type is *http:api*.

$lib.cortex.httpapi.del(iden)

Delete an Extended HTTP API endpoint.

Args:

iden (string): The iden of the API to delete.

Returns:

The return type is `null`.

$lib.cortex.httpapi.get(iden)

Get an Extended `http:api` object.

Args:

iden (string): The iden of the API to retrieve.

Returns:

The `http:api` object. The return type is *http:api*.

$lib.cortex.httpapi.getByPath(path)

Get an Extended `http:api` object by path.

Notes:
The path argument is evaluated as a regular expression input, and will be used to get the first HTTP API handler whose path value has a match.

Args:
path (string): Path to use to retrieve an object.

Returns:
The `http:api` object or `$lib.null` if there is no match. The return type may be one of the following: *http:api*, `null`.

$lib.cortex.httpapi.index(iden, index=(0))

Set the index for a given Extended HTTP API.

Args:
iden (string): The iden of the API to modify.

index (int): The new index of the API. Uses zero based indexing.

Returns:
The new index location of the API. The return type is `int`.

$lib.cortex.httpapi.list()

Get all the Extended HTTP APIs on the Cortex

Returns:
A list of `http:api` objects The return type is *list*.

$lib.cortex.httpapi.response(requestinfo)

Make a response object. Used by API handlers automatically.

Args:
requestinfo (dict): Request info dictionary. This is an opaque data structure which may change.

Returns:
The return type is *http:api:request*.

13.1.20 $lib.cron

A Storm Library for interacting with Cron Jobs in the Cortex.

$lib.cron.add(**kwargs)

Add a recurring Cron Job to the Cortex.

Args:
**kwargs (any): Key-value parameters used to add the cron job.

Returns:
The new Cron Job. The return type is *cronjob*.

$lib.cron.at(**kwargs)

Add a non-recurring Cron Job to the Cortex.

Args:
**kwargs (any): Key-value parameters used to add the cron job.

Returns:
The new Cron Job. The return type is *cronjob*.

$lib.cron.del(prefix)

Delete a CronJob from the Cortex.

Args:
prefix (str): A prefix to match in order to identify a cron job to delete. Only a single matching prefix will be deleted.

Returns:
The return type is `null`.

$lib.cron.disable(prefix)

Disable a CronJob in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a cron job to disable. Only a single matching prefix will be disabled.

Returns:
The iden of the CronJob which was disabled. The return type is *str*.

$lib.cron.enable(prefix)

Enable a CronJob in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a cron job to enable. Only a single matching prefix will be enabled.

Returns:
The iden of the CronJob which was enabled. The return type is *str*.

$lib.cron.get(prefix)

Get a CronJob in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a cron job to get. Only a single matching prefix will be retrieved.

Returns:
The requested cron job. The return type is *cronjob*.

$lib.cron.list()

List CronJobs in the Cortex.

Returns:
A list of `cronjob` objects. The return type is *list*.

$lib.cron.mod(prefix, query)

Modify the Storm query for a CronJob in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a cron job to modify. Only a single matching prefix will be modified.

query: The new Storm query for the Cron Job. The input type may be one of the following: `str`, `query`.

Returns:
The iden of the CronJob which was modified. The return type is *str*.

$lib.cron.move(prefix, view)

Move a cron job to a new view.

Args:
prefix (str): A prefix to match in order to identify a cron job to move. Only a single matching prefix will be modified.

view (str): The iden of the view to move the CrobJob to

Returns:
The iden of the CronJob which was moved. The return type is *str*.

13.1.21 $lib.crypto.coin.ethereum

A Storm library which implements helpers for Ethereum.

$lib.crypto.coin.ethereum.eip55(addr)

Convert an Ethereum address to a checksummed address.

Args:
addr (str): The Ethereum address to be converted.

Returns:
A list of (<bool>, <addr>) for status and checksummed address. The return type is *list*.

13.1.22 $lib.crypto.hashes

A Storm Library for hashing bytes

$lib.crypto.hashes.md5(byts)

Retrieve an MD5 hash of a byte string.

Args:
byts (bytes): The bytes to hash.

Returns:
The hex digest of the MD5 hash of the input bytes. The return type is *str*.

$lib.crypto.hashes.sha1(byts)

Retrieve a SHA1 hash of a byte string.

Args:
byts (bytes): The bytes to hash.

Returns:
The hex digest of the SHA1 hash of the input bytes. The return type is *str*.

$lib.crypto.hashes.sha256(byts)

Retrieve a SHA256 hash of a byte string.

Args:
byts (bytes): The bytes to hash.

Returns:
The hex digest of the SHA256 hash of the input bytes. The return type is *str*.

$lib.crypto.hashes.sha512(byts)

Retrieve a SHA512 hash of a byte string.

Args:
byts (bytes): The bytes to hash.

Returns:
The hex digest of the SHA512 hash of the input bytes. The return type is *str*.

13.1.23 $lib.crypto.hmac

A Storm library for computing RFC2104 HMAC values.

$lib.crypto.hmac.digest(key, mesg, alg=sha256)

Compute the digest value of a message using RFC2104 HMAC.

Examples:

Compute the HMAC-SHA256 digest for a message with a secret key:

```
$digest = $lib.crypto.hmac.digest(key=$secretKey.encode(), mesg=$mesg.encode())
```

Args:

key (bytes): The key to use for the HMAC calculation.

mesg (bytes): The message to use for the HMAC calculation.

alg (str): The digest algorithm to use.

Returns:

The binary digest of the HMAC value. The return type is *bytes*.

13.1.24 $lib.csv

A Storm Library for interacting with csvtool.

$lib.csv.emit(*args, table=$lib.null)

Emit a `csv:row` event to the Storm runtime for the given args.

Args:

*args (any): Items which are emitted as a `csv:row` event.

table (str): The name of the table to emit data too. Optional.

Returns:

The return type is `null`.

13.1.25 $lib.dict

A Storm Library for interacting with dictionaries.

$lib.dict.keys(valu)

Retrieve a list of keys in the specified dictionary.

Args:

valu (dict): The dictionary to operate on.

Returns:

List of keys in the specified dictionary. The return type is *list*.

$lib.dict.pop(valu, key, default=$lib.undef)

Remove specified key and return the corresponding value.

Args:

valu (dict): The dictionary to operate on.

key (str): The key name of the value to pop.

default (any): Optional default value to return if the key does not exist in the dictionary.

Returns:

The popped value. The return type is `any`.

$lib.dict.update(valu, other)

Update the specified dictionary with keys/values from another dictionary.

Args:

valu (dict): The target dictionary (update to).

other (dict): The source dictionary (update from).

Returns:

The return type is `null`.

$lib.dict.values(valu)

Retrieve a list of values in the specified dictionary.

Args:

valu (dict): The dictionary to operate on.

Returns:

List of values in the specified dictionary. The return type is *list*.

13.1.26 $lib.dmon

A Storm Library for interacting with StormDmons.

$lib.dmon.add(text, name=noname, ddef=$lib.null)

Add a Storm Dmon to the Cortex.

Examples:

Add a dmon that executes a query:

```
$lib.dmon.add(${ myquery }, name='example dmon')
```

Args:

text: The Storm query to execute in the Dmon loop. The input type may be one of the following: `str`, `storm:query`.

name (str): The name of the Dmon.

ddef (dict): Additional daemon definition fields.

Returns:
The iden of the newly created Storm Dmon. The return type is *str*.

$lib.dmon.bump(iden)

Restart the Dmon.

Args:
iden (str): The GUID of the dmon to restart.

Returns:
True if the Dmon is restarted; False if the iden does not exist. The return type is *boolean*.

$lib.dmon.del(iden)

Delete a Storm Dmon by iden.

Args:
iden (str): The iden of the Storm Dmon to delete.

Returns:
The return type is `null`.

$lib.dmon.get(iden)

Get a Storm Dmon definition by iden.

Args:
iden (str): The iden of the Storm Dmon to get.

Returns:
A Storm Dmon definition dict. The return type is *dict*.

$lib.dmon.list()

Get a list of Storm Dmons.

Returns:
A list of Storm Dmon definitions. The return type is *list*.

$lib.dmon.log(iden)

Get the messages from a Storm Dmon.

Args:
iden (str): The iden of the Storm Dmon to get logs for.

Returns:
A list of messages from the StormDmon. The return type is *list*.

$lib.dmon.start(iden)

Start a storm dmon.

Args:
iden (str): The GUID of the dmon to start.

Returns:
$lib.true unless the dmon does not exist or was already started. The return type is *boolean*.

$lib.dmon.stop(iden)

Stop a Storm Dmon.

Args:
iden (str): The GUID of the Dmon to stop.

Returns:
$lib.true unless the dmon does not exist or was already stopped. The return type is *boolean*.

13.1.27 $lib.export

A Storm Library for exporting data.

$lib.export.toaxon(query, opts=$lib.null)

Run a query as an export (fully resolving relationships between nodes in the output set) and save the resulting stream of packed nodes to the axon.

Args:
query (str): A query to run as an export.

opts (dict): Storm runtime query option params.

Returns:
Returns a tuple of (size, sha256). The return type is *list*.

13.1.28 $lib.feed

A Storm Library for interacting with Cortex feed functions.

$lib.feed.genr(name, data)

Yield nodes being added to the graph by adding data with a given ingest type.

Notes:
This is using the Runtimes's Snap to call addFeedNodes(). This only yields nodes if the feed function yields nodes. If the generator is not entirely consumed there is no guarantee that all of the nodes which should be made by the feed function will be made.

Args:
name (str): Name of the ingest function to send data too.

data (prim): Data to send to the ingest function.

Yields:
Yields Nodes as they are created by the ingest function. The return type is *node*.

$lib.feed.ingest(name, data)

Add nodes to the graph with a given ingest type.

Notes:
This is using the Runtimes's Snap to call addFeedData(), after setting the snap.strict mode to False. This will cause node creation and property setting to produce warning messages, instead of causing the Storm Runtime to be torn down.

Args:
name (str): Name of the ingest function to send data too.

data (prim): Data to send to the ingest function.

Returns:
The return type is `null`.

$lib.feed.list()

Get a list of feed functions.

Returns:
A list of feed functions. The return type is *list*.

13.1.29 $lib.gen

A Storm Library for secondary property based deconfliction.

$lib.gen.campaign(name, reporter)

Returns an ou:campaign node based on the campaign and reporter names, adding the node if it does not exist.

Args:
name (str): The reported name of the campaign.

reporter (str): The name of the organization which reported the campaign.

Returns:
An ou:campaign node. The return type is *node*.

$lib.gen.geoPlaceByName(name)

Returns a geo:place node by name, adding the node if it does not exist.

Args:
name (str): The name of the place.

Returns:
A geo:place node with the given name. The return type is *node*.

$lib.gen.industryByName(name)

Returns an ou:industry by name, adding the node if it does not exist.

Args:
name (str): The name of the industry.

Returns:
An ou:industry node with the given name. The return type is *node*.

$lib.gen.itAvScanResultByTarget(form, value, signame, scanner=$lib.null, time=$lib.null, try=$lib.false)

Returns an it:av:scan:result node by deconflicting with a target and signature name, adding the node if it does not exist.

Args:
form (str): The target form.

value (str): The target value.

signame (str): The signature name.

scanner (str): An optional scanner software name to include in deconfliction.

time (time): An optional time when the scan was run to include in the deconfliction.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
An it:av:scan:result node. The return type is *node*.

$lib.gen.langByCode(name, try=$lib.false)

Returns a lang:language node by language code, adding the node if it does not exist.

Args:
name (str): The language code for the language.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
A lang:language node with the given code. The return type is *node*.

$lib.gen.langByName(name)

Returns a lang:language node by name, adding the node if it does not exist.

Args:
name (str): The name of the language.

Returns:
A lang:language node with the given name. The return type is *node*.

$lib.gen.newsByUrl(url, try=$lib.false)

Returns a media:news node by URL, adding the node if it does not exist.

Args:
url (inet:url): The URL where the news is published.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
A media:news node with the given URL. The return type is *node*.

$lib.gen.orgByFqdn(fqdn, try=$lib.false)

Returns an ou:org node by FQDN, adding the node if it does not exist.

Args:
fqdn (str): The FQDN of the org.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
An ou:org node with the given FQDN. The return type is *node*.

$lib.gen.orgByName(name)

Returns an ou:org by name, adding the node if it does not exist.

Args:
name (str): The name of the org.

Returns:
An ou:org node with the given name. The return type is *node*.

$lib.gen.orgHqByName(name)

Returns a ps:contact node for the ou:org, adding the node if it does not exist.

Args:
name (str): The name of the org.

Returns:
A ps:contact node for the ou:org with the given name. The return type is *node*.

$lib.gen.polCountryByIso2(iso2, try=$lib.false)

Returns a pol:country node by deconflicting the :iso2 property.

Args:
iso2 (str): The pol:country:iso2 property.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
A pol:country node. The return type is *node*.

$lib.gen.psContactByEmail(type, email, try=$lib.false)

Returns a ps:contact by deconflicting the type and email address.

Args:
type (str): The ps:contact:type property.

email (str): The ps:contact:email property.

try (boolean): Type normalization will fail silently instead of raising an exception.

Returns:
A ps:contact node. The return type is *node*.

$lib.gen.riskThreat(name, reporter)

Returns a risk:threat node based on the threat and reporter names, adding the node if it does not exist.

Args:
name (str): The reported name of the threat cluster.

reporter (str): The name of the organization which reported the threat cluster.

Returns:
A risk:threat node. The return type is *node*.

$lib.gen.riskToolSoftware(name, reporter)

Returns a risk:tool:software node based on the tool and reporter names, adding the node if it does not exist.

Args:
name (str): The reported name of the tool.

reporter (str): The name of the organization which reported the tool.

Returns:
A risk:tool:software node. The return type is *node*.

$lib.gen.softByName(name)

Returns it:prod:soft node by name, adding the node if it does not exist.

Args:
name (str): The name of the software.

Returns:
An it:prod:soft node with the given name. The return type is *node*.

$lib.gen.vulnByCve(cve, try=$lib.false, reporter=$lib.null)

Returns risk:vuln node by CVE and reporter, adding the node if it does not exist.

Args:
cve (str): The CVE id.

try (boolean): Type normalization will fail silently instead of raising an exception.

reporter (str): The name of the organization which reported the vulnerability.

Returns:
A risk:vuln node with the given CVE. The return type is *node*.

13.1.30 $lib.gis

A Storm library which implements helpers for earth based geospatial calculations.

$lib.gis.bbox(lon, lat, dist)

Calculate a min/max bounding box for the specified circle.

Args:
lon (float): The longitude in degrees.

lat (float): The latitude in degrees.

dist (int): A distance in geo:dist base units (mm).

Returns:
A tuple of (lonmin, lonmax, latmin, latmax). The return type is *list*.

13.1.31 $lib.globals

A Storm Library for interacting with global variables which are persistent across the Cortex.

$lib.globals.get(name, default=$lib.null)

Get a Cortex global variables.

Args:
name (str): Name of the variable.

default (prim): Default value to return if the variable is not set.

Returns:
The variable value. The return type is `prim`.

$lib.globals.list()

Get a list of variable names and values.

Returns:
A list of tuples with variable names and values that the user can access. The return type is *list*.

$lib.globals.pop(name, default=$lib.null)

Delete a variable value from the Cortex.

Args:
name (str): Name of the variable.

default (prim): Default value to return if the variable is not set.

Returns:
The variable value. The return type is `prim`.

$lib.globals.set(name, valu)

Set a variable value in the Cortex.

Args:
name (str): The name of the variable to set.

valu (prim): The value to set.

Returns:
The variable value. The return type is `prim`.

13.1.32 $lib.graph

A Storm Library for interacting with graph projections in the Cortex.

$lib.graph.activate(iden)

Set the graph projection to use for the top level Storm Runtime.

Args:
iden (str): The iden of the graph projection to use.

Returns:
The return type is `null`.

$lib.graph.add(gdef)

Add a graph projection to the Cortex.

Example:
Add a graph projection named "Test Projection":

```
$rules = ({
    "name": "Test Projection",
    "desc": "My test projection",
    "degrees": 2,
    "pivots": ["-> meta:seen"],
    "filters": ["-#nope"],
    "forms": {
        "inet:fqdn": {
            "pivots": ["<- *", "-> *"],
            "filters": ["-inet:fqdn:issuffix=1"]
        },
        "*": {
            "pivots": ["-> #"],
        }
    }
})
$lib.graph.add($rules)
```

Args:
gdef (dict): A graph projection definition.

Returns:
The return type is `null`.

$lib.graph.del(iden)

Delete a graph projection from the Cortex.

Args:
iden (str): The iden of the graph projection to delete.

Returns:
The return type is `null`.

$lib.graph.get(iden=$lib.null)

Get a graph projection definition from the Cortex.

Args:
iden (str): The iden of the graph projection to get. If not specified, returns the current graph projection.

Returns:
A graph projection definition, or None if no iden was specified and there is currently no graph projection set. The return type is *dict*.

$lib.graph.grant(gden, scope, iden, level)

Modify permissions granted to users/roles on a graph projection.

Args:

gden (str): Iden of the graph projection to modify.

scope (str): The scope, either "users" or "roles".

iden (str): The user/role iden depending on scope.

level (int): The permission level number.

Returns:

The return type is `null`.

$lib.graph.list()

List the graph projections available in the Cortex.

Returns:

A list of graph projection definitions. The return type is *list*.

$lib.graph.mod(iden, info)

Modify user editable properties of a graph projection.

Args:

iden (str): The iden of the graph projection to modify.

info (dict): A dictionary of the properties to edit.

Returns:

The return type is `null`.

13.1.33 $lib.hex

A Storm library which implements helpers for hexadecimal encoded strings.

$lib.hex.decode(valu)

Decode a hexadecimal string into bytes.

Args:

valu (str): The hex string to be decoded into bytes.

Returns:

The decoded bytes. The return type is *bytes*.

$lib.hex.encode(valu)

Encode bytes into a hexadecimal string.

Args:
valu (bytes): The bytes to be encoded into a hex string.

Returns:
The hex encoded string. The return type is *str*.

$lib.hex.fromint(valu, length, signed=$lib.false)

Convert an integer to a big endian hexadecimal string.

Args:
valu (int): The integer to be converted.

length (int): The number of bytes to use to represent the integer.

signed (bool): If true, convert as a signed value.

Returns:
The resulting hex string. The return type is *str*.

$lib.hex.signext(valu, length)

Sign extension pad a hexadecimal encoded signed integer.

Args:
valu (str): The hex string to pad.

length (int): The number of characters to pad the string to.

Returns:
The sign extended hex string. The return type is *str*.

$lib.hex.toint(valu, signed=$lib.false)

Convert a big endian hexadecimal string to an integer.

Args:
valu (str): The hex string to be converted.

signed (bool): If true, convert to a signed integer.

Returns:
The resulting integer. The return type is `int`.

$lib.hex.trimext(valu)

Trim sign extension bytes from a hexadecimal encoded signed integer.

Args:
valu (str): The hex string to trim.

Returns:
The trimmed hex string. The return type is *str*.

13.1.34 $lib.inet.http

A Storm Library exposing an HTTP client API.

For APIs that accept an ssl_opts argument, the dictionary may contain the following values:

```
{
    'verify': <bool> - Perform SSL/TLS verification. Is overridden by the ssl_verify␣
↪argument.
    'client_cert': <str> - PEM encoded full chain certificate for use in mTLS.
    'client_key': <str> - PEM encoded key for use in mTLS. Alternatively, can be␣
↪included in client_cert.
}
```

$lib.inet.http.codereason(code)

Get the reason phrase for an HTTP status code.

Examples:
Get the reason for a 404 status code:

```
$str=$lib.inet.http.codereason(404)
```

Args:
code (int): The HTTP status code.

Returns:
The reason phrase for the status code. The return type is *str*.

$lib.inet.http.connect(url, headers=$lib.null, ssl_verify=$lib.true, timeout=(300), params=$lib.null, proxy=$lib.null, ssl_opts=$lib.null)

Connect a web socket to tx/rx JSON messages.

Args:
url (str): The URL to retrieve.

headers (dict): HTTP headers to send with the request.

ssl_verify (boolean): Perform SSL/TLS verification.

timeout (int): Total timeout for the request in seconds.

params (dict): Optional parameters which may be passed to the connection request.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.inet.http help for additional details.

Returns:
A websocket object. The return type is *inet:http:socket*.

$lib.inet.http.get(url, headers=$lib.null, ssl_verify=$lib.true, params=$lib.null, timeout=(300), allow_redirects=$lib.true, proxy=$lib.null, ssl_opts=$lib.null)

Get the contents of a given URL.

Args:
url (str): The URL to retrieve.

headers (dict): HTTP headers to send with the request.

ssl_verify (boolean): Perform SSL/TLS verification.

params (dict): Optional parameters which may be passed to the request.

timeout (int): Total timeout for the request in seconds.

allow_redirects (bool): If set to false, do not follow redirects.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.inet.http help for additional details.

Returns:
The response object. The return type is *inet:http:resp*.

$lib.inet.http.head(url, headers=$lib.null, ssl_verify=$lib.true, params=$lib.null, timeout=(300), allow_redirects=$lib.false, proxy=$lib.null, ssl_opts=$lib.null)

Get the HEAD response for a URL.

Args:
url (str): The URL to retrieve.

headers (dict): HTTP headers to send with the request.

ssl_verify (boolean): Perform SSL/TLS verification.

params (dict): Optional parameters which may be passed to the request.

timeout (int): Total timeout for the request in seconds.

allow_redirects (bool): If set to true, follow redirects.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.inet.http help for additional details.

Returns:
The response object. The return type is *inet:http:resp*.

$lib.inet.http.post(url, headers=$lib.null, json=$lib.null, body=$lib.null, ssl_verify=$lib.true, params=$lib.null, timeout=(300), allow_redirects=$lib.true, fields=$lib.null, proxy=$lib.null, ssl_opts=$lib.null)

Post data to a given URL.

Args:

url (str): The URL to post to.

headers (dict): HTTP headers to send with the request.

json (prim): The data to post, as JSON object.

body (bytes): The data to post, as binary object.

ssl_verify (boolean): Perform SSL/TLS verification.

params (dict): Optional parameters which may be passed to the request.

timeout (int): Total timeout for the request in seconds.

allow_redirects (bool): If set to false, do not follow redirects.

fields (list): A list of info dictionaries containing the name, value or sha256, and additional parameters for fields to post, as multipart/form-data. If a sha256 is specified, the request will be sent from the axon and the corresponding file will be uploaded as the value for the field.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.inet.http help for additional details.

Returns:

The response object. The return type is *inet:http:resp*.

$lib.inet.http.request(meth, url, headers=$lib.null, json=$lib.null, body=$lib.null, ssl_verify=$lib.true, params=$lib.null, timeout=(300), allow_redirects=$lib.true, fields=$lib.null, proxy=$lib.null, ssl_opts=$lib.null)

Make an HTTP request using the given HTTP method to the url.

Args:

meth (str): The HTTP method. (ex. PUT)

url (str): The URL to send the request to.

headers (dict): HTTP headers to send with the request.

json (prim): The data to include in the body, as JSON object.

body (bytes): The data to include in the body, as binary object.

ssl_verify (boolean): Perform SSL/TLS verification.

params (dict): Optional parameters which may be passed to the request.

timeout (int): Total timeout for the request in seconds.

allow_redirects (bool): If set to false, do not follow redirects.

fields (list): A list of info dictionaries containing the name, value or sha256, and additional parameters for fields to post, as multipart/form-data. If a sha256 is specified, the request will be sent from the axon and the corresponding file will be uploaded as the value for the field.

proxy: Set to a proxy URL string or $lib.false to disable proxy use. The input type may be one of the following: `bool`, `null`, `str`.

ssl_opts (dict): Optional SSL/TLS options. See $lib.inet.http help for additional details.

Returns:
The response object. The return type is *inet:http:resp*.

$lib.inet.http.urldecode(text)

Urldecode a text string.

This will replace %xx escape characters with the special characters they represent and replace plus signs with spaces.

Examples:
Urlencode a string:

```
$str=$lib.inet.http.urldecode("http%3A%2F%2Fgo+ogle.com")
```

Args:
text (str): The text string.

Returns:
The urldecoded string. The return type is *str*.

$lib.inet.http.urlencode(text)

Urlencode a text string.

This will replace special characters in a string using the %xx escape and replace spaces with plus signs.

Examples:
Urlencode a string:

```
$str=$lib.inet.http.urlencode("http://google.com")
```

Args:
text (str): The text string.

Returns:
The urlencoded string. The return type is *str*.

13.1.35 $lib.inet.http.oauth.v1

A Storm library to handle OAuth v1 authentication.

$lib.inet.http.oauth.v1.client(ckey, csecret, atoken, asecret, sigtype=QUERY)

Initialize an OAuthV1 Client to use for signing/authentication.

Args:

ckey (str): The OAuthV1 Consumer Key to store and use for signing requests.

csecret (str): The OAuthV1 Consumer Secret used to sign requests.

atoken (str): The OAuthV1 Access Token (or resource owner key) to use to sign requests.)

asecret (str): The OAuthV1 Access Token Secret (or resource owner secret) to use to sign requests.

sigtype (str): Where to populate the signature (in the HTTP body, in the query parameters, or in the header)

Returns:

An OAuthV1 client to be used to sign requests. The return type is *inet:http:oauth:v1:client*.

13.1.36 $lib.inet.http.oauth.v2

A Storm library for managing OAuth V2 clients.

$lib.inet.http.oauth.v2.addProvider(conf)

Add a new provider configuration.

Example:

Add a new provider which uses the authorization code flow:

```
$iden = $lib.guid(example, provider, oauth)
$conf = ({
    "iden": $iden,
    "name": "example_provider",
    "client_id": "yourclientid",
    "client_secret": "yourclientsecret",
    "scope": "first_scope second_scope",
    "auth_uri": "https://provider.com/auth",
    "token_uri": "https://provider.com/token",
    "redirect_uri": "https://local.redirect.com/oauth",
})

// Optionally enable PKCE
$conf.extensions = ({"pkce": $lib.true})

// Optionally disable SSL verification
$conf.ssl_verify = $lib.false

// Optionally provide additional key-val parameters
// to include when calling the auth URI
$conf.extra_auth_params = ({"customparam": "foo"})

$lib.inet.http.oauth.v2.addProvider($conf)
```

Args:

conf (dict): A provider configuration.

Returns:
The return type is `null`.

$lib.inet.http.oauth.v2.clearUserAccessToken(iden)

Clear the stored refresh data for the current user's provider access token.

Args:
iden (str): The provider iden.

Returns:
The existing token state data or None if it did not exist. The return type is *dict*.

$lib.inet.http.oauth.v2.delProvider(iden)

Delete a provider configuration.

Args:
iden (str): The provider iden.

Returns:
The deleted provider configuration or None if it does not exist. The return type is *dict*.

$lib.inet.http.oauth.v2.getProvider(iden)

Get a provider configuration

Args:
iden (str): The provider iden.

Returns:
The provider configuration or None if it does not exist. The return type is *dict*.

$lib.inet.http.oauth.v2.getUserAccessToken(iden)

Get the provider access token for the current user.

Example:

Retrieve the token and handle needing an auth code:

```
$provideriden = $lib.globals.get("oauth:myprovider")

($ok, $data) = $lib.inet.http.oauth.v2.getUserAccessToken($provideriden)

if $ok {
    // $data is the token to be used in a request
else {
    // $data is a message stating why the token is not available
    // caller should now handle retrieving a new auth code for the user
}
```

Args:
iden (str): The provider iden.

Returns:
List of (<bool>, <token/mesg>) for status and data. The return type is *list*.

$lib.inet.http.oauth.v2.listProviders()

List provider configurations

Returns:
List of (iden, conf) tuples. The return type is *list*.

$lib.inet.http.oauth.v2.setUserAuthCode(iden, authcode, code_verifier=$lib.null)

Set the auth code for the current user.

Args:
iden (str): The provider iden.

authcode (str): The auth code for the user.

code_verifier (str): Optional PKCE code verifier.

Returns:
The return type is `null`.

13.1.37 $lib.inet.imap

A Storm library to connect to an IMAP server.

$lib.inet.imap.connect(host, port=(993), timeout=(30), ssl=$lib.true)

Open a connection to an IMAP server.

This method will wait for a "hello" response from the server before returning the `inet:imap:server` instance.

Args:
host (str): The IMAP hostname.

port (int): The IMAP server port.

timeout (int): The time to wait for all commands on the server to execute.

ssl (bool): Use SSL to connect to the IMAP server.

Returns:
A new `inet:imap:server` instance. The return type is *inet:imap:server*.

13.1.38 $lib.inet.ipv6

A Storm Library for providing ipv6 helpers.

$lib.inet.ipv6.expand(valu)

Convert a IPv6 address to its expanded form.'

Notes:
The expanded form is also sometimes called the "long form" address.

Examples:
Expand a ipv6 address to its long form:

```
$expandedvalu = $lib.inet.ipv6.expand('2001:4860:4860::8888')
```

Args:
valu (str): IPv6 Address to expand

Returns:
The expanded form. The return type is *str*.

13.1.39 $lib.inet.smtp

A Storm Library for sending email messages via SMTP.

$lib.inet.smtp.message()

Construct a new email message.

Returns:
The newly constructed inet:smtp:message. The return type is *inet:smtp:message*.

13.1.40 $lib.inet.whois

A Storm Library for providing a consistent way to generate guids for WHOIS / Registration Data in Storm.

$lib.inet.whois.guid(props, form)

Provides standard patterns for creating guids for certain inet:whois forms.

Raises:
StormRuntimeError: If form is not supported in this method.

Args:
props (dict): Dictionary of properties used to create the form.

form (str): The `inet:whois` form to create the guid for.

Returns:
A guid for creating a the node for. The return type is *str*.

13.1.41 $lib.infosec.cvss

A Storm library which implements CVSS score calculations.

$lib.infosec.cvss.calculate(node, save=$lib.true, vers=3.1)

Calculate the CVSS score values for an input risk:vuln node.

Args:
node (node): A risk:vuln node from the Storm runtime.

save (boolean): If true, save the computed scores to the node properties.

vers (str): The version of CVSS calculations to execute.

Returns:
A dictionary containing the computed score and subscores. The return type is *dict*.

$lib.infosec.cvss.calculateFromProps(props, vers=3.1)

Calculate the CVSS score values from a props dict.

Args:
props (dict): A props dictionary.

vers (str): The version of CVSS calculations to execute.

Returns:
A dictionary containing the computed score and subscores. The return type is *dict*.

$lib.infosec.cvss.saveVectToNode(node, text)

Parse a CVSS v3.1 vector and record properties on a risk:vuln node.

Args:
node (node): A risk:vuln node to record the CVSS properties on.

text (str): A CVSS vector string.

Returns:
The return type is `null`.

$lib.infosec.cvss.vectToProps(text)

Parse a CVSS v3.1 vector and return a dictionary of risk:vuln props.

Args:
text (str): A CVSS vector string.

Returns:
A dictionary of risk:vuln secondary props. The return type is *dict*.

$lib.infosec.cvss.vectToScore(vect, vers=$lib.null)

Compute CVSS scores from a vector string.

Takes a CVSS vector string, attempts to automatically detect the version (defaults to CVSS3.1 if it cannot), and calculates the base, temporal, and environmental scores.

Raises:

- BadArg: An invalid *vers* string is provided
- BadDataValu: The vector string is invalid in some way. Possible reasons are malformed string, duplicated metrics, missing mandatory metrics, and invalid metric values.

Args:

vect (str):
A valid CVSS vector string.

The following examples are valid formats:

- CVSS 2 with version: *CVSS2#AV:L/AC:L/Au:M/C:P/I:C/A:N*
- CVSS 2 with parentheses: *(AV:L/AC:L/Au:M/C:P/I:C/A:N)*
- CVSS 2 without parentheses: *AV:L/AC:L/Au:M/C:P/I:C/A:N*
- CVSS 3.0 with version: *CVSS:3.0/AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L*
- CVSS 3.1 with version: *CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L*
- CVSS 3.0/3.1 with parentheses: *(AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L)*
- CVSS 3.0/3.1 without parentheses: *AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L*

vers (str):
A valid version string or None to autodetect the version from the vector string. Accepted values are: 2, 3.0, 3.1, None.

Returns:

A dictionary with the detected version, base score, temporal score, environmental score, overall score, and normalized vector string. The normalized vector string will have metrics ordered in specification order and metrics with undefined values will be removed. Example:

```
{
    'version': '3.1',
    'score': 4.3,
    'base': 5.0,
    'temporal': 4.4,
    'environmental': 4.3,
    'normalized': 'AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:L'
}
```

The return type is *dict*.

13.1.42 $lib.infosec.mitre.attack.flow

A Storm library which implements modeling MITRE ATT&CK Flow diagrams.

$lib.infosec.mitre.attack.flow.ingest(flow)

Ingest a MITRE ATT&CK Flow diagram in JSON format.

Args:
flow (data): The JSON data to ingest.

Returns:
The it:mitre:attack:flow node representing the ingested attack flow diagram. The return type may be one of the following: *node*, `null`.

$lib.infosec.mitre.attack.flow.norm(flow)

Normalize a MITRE ATT&CK Flow diagram in JSON format.

Args:
flow (data): The MITRE ATT&CK Flow diagram in JSON format to normalize (flatten and sort).

Returns:
The normalized MITRE ATT&CK Flow diagram. The return type is *dict*.

13.1.43 $lib.iters

A Storm library for providing iterator helpers.

$lib.iters.enum(genr)

Yield (<indx>, <item>) tuples from an iterable or generator.

Args:
genr (iter): An iterable or generator.

Yields:
Yields (<indx>, <item>) tuples. The return type is *list*.

$lib.iters.zip(*args)

Yield tuples created by iterating multiple iterables in parallel.

Args:
*args (iter): Iterables or generators.

Yields:
Yields tuples with an item from each iterable or generator. The return type is *list*.

13.1.44 $lib.json

A Storm Library for interacting with Json data.

$lib.json.load(text)

Parse a JSON string and return the deserialized data.

Args:
text (str): The string to be deserialized.

Returns:
The JSON deserialized object. The return type is `prim`.

$lib.json.save(item)

Save an object as a JSON string.

Args:
item (any): The item to be serialized as a JSON string.

Returns:
The JSON serialized object. The return type is *str*.

$lib.json.schema(schema, use_default=$lib.true)

Get a JS schema validation object.

Args:
schema (dict): The JsonSchema to use.

use_default (boolean): Whether to insert default schema values into the validated data structure.

Returns:
A validation object that can be used to validate data structures. The return type is *json:schema*.

13.1.45 $lib.jsonstor

Implements cortex JSON storage.

$lib.jsonstor.cachedel(path, key)

Remove cached data set with cacheset.

Args:
path (str|list): The base path to use for the cache key.

key (prim): The value to use for the GUID cache key.

Returns:
True if the del operation was successful. The return type is *boolean*.

$lib.jsonstor.cacheget(path, key, asof=now, envl=$lib.false)

Retrieve data stored with cacheset() if it was stored more recently than the asof argument.

Args:
path (str|list): The base path to use for the cache key.

key (prim): The value to use for the GUID cache key.

asof (time): The max cache age.

envl (boolean): Return the full cache envelope.

Returns:
The cached value (or envelope) or null. The return type is `prim`.

$lib.jsonstor.cacheset(path, key, valu)

Set cache data with an envelope that tracks time for cacheget() use.

Args:
path (str|list): The base path to use for the cache key.

key (prim): The value to use for the GUID cache key.

valu (prim): The data to store.

Returns:
The cached asof time and path. The return type is *dict*.

$lib.jsonstor.del(path, prop=$lib.null)

Delete a stored JSON object or object.

Args:
path (str|list): A path string or list of path parts.

prop (str|list): A property name or list of name parts.

Returns:
True if the del operation was successful. The return type is *boolean*.

$lib.jsonstor.get(path, prop=$lib.null)

Return a stored JSON object or object property.

Args:
path (str|list): A path string or list of path parts.

prop (str|list): A property name or list of name parts.

Returns:
The previously stored value or $lib.null The return type is `prim`.

$lib.jsonstor.iter(path=$lib.null)

Yield (<path>, <valu>) tuples for the JSON objects.

Args:
path (str|list): A path string or list of path parts.

Yields:
(<path>, <item>) tuples. The return type is *list*.

$lib.jsonstor.set(path, valu, prop=$lib.null)

Set a JSON object or object property.

Args:
path (str|list): A path string or list of path elements.

valu (prim): The value to set as the JSON object or object property.

prop (str|list): A property name or list of name parts.

Returns:
True if the set operation was successful. The return type is *boolean*.

13.1.46 $lib.layer

A Storm Library for interacting with Layers in the Cortex.

$lib.layer.add(ldef=$lib.null)

Add a layer to the Cortex.

Args:
ldef (dict): The layer definition dictionary.

Returns:
A `layer` object representing the new layer. The return type is *layer*.

$lib.layer.del(iden)

Delete a layer from the Cortex.

Args:
iden (str): The iden of the layer to delete.

Returns:
The return type is `null`.

$lib.layer.get(iden=$lib.null)

Get a Layer from the Cortex.

Args:
iden (str): The iden of the layer to get. If not set, this defaults to the top layer of the current View.

Returns:
The storm layer object. The return type is *layer*.

$lib.layer.list()

List the layers in a Cortex

Returns:
List of `layer` objects. The return type is *list*.

13.1.47 $lib.lift

A Storm Library for interacting with lift helpers.

$lib.lift.byNodeData(name)

Lift nodes which have a given nodedata name set on them.

Args:
name (str): The name to of the nodedata key to lift by.

Yields:
Yields nodes to the pipeline. This must be used in conjunction with the `yield` keyword. The return type is *node*.

13.1.48 $lib.log

A Storm library which implements server side logging. These messages are logged to the `synapse.storm.log` logger.

$lib.log.debug(mesg, extra=$lib.null)

Log a message to the Cortex at the debug log level.

Notes:
This requires the `storm.lib.log.debug` permission to use.

Examples:
Log a debug message:

```
$lib.log.debug('I am a debug message!')
```

Log a debug message with extra information:

```
$lib.log.debug('Extra information included here.', extra=({"key": $valu}))
```

Args:
mesg (str): The message to log.

extra (dict): Extra key / value pairs to include when structured logging is enabled on the Cortex.

Returns:
The return type is `null`.

$lib.log.error(mesg, extra=$lib.null)

Log a message to the Cortex at the error log level.

Notes:
This requires the `storm.lib.log.error` permission to use.

Examples:
Log an error message:

```
$lib.log.error('I am a error message!')
```

Log an error message with extra information:

```
$lib.log.error('Extra information included here.', extra=({"key": $valu}))
```

Args:
mesg (str): The message to log.

extra (dict): Extra key / value pairs to include when structured logging is enabled on the Cortex.

Returns:
The return type is `null`.

$lib.log.info(mesg, extra=$lib.null)

Log a message to the Cortex at the info log level.

Notes:
This requires the `storm.lib.log.info` permission to use.

Examples:
Log an info message:

```
$lib.log.info('I am a info message!')
```

Log an info message with extra information:

```
$lib.log.info('Extra information included here.', extra=({"key": $valu}))
```

Args:
mesg (str): The message to log.

extra (dict): Extra key / value pairs to include when structured logging is enabled on the Cortex.

Returns:
The return type is `null`.

$lib.log.warning(mesg, extra=$lib.null)

Log a message to the Cortex at the warning log level.

Notes:
This requires the `storm.lib.log.warning` permission to use.

Examples:
Log a warning message:

```
$lib.log.warning('I am a warning message!')
```

Log a warning message with extra information:

```
$lib.log.warning('Extra information included here.', extra=({"key": $valu}))
```

Args:
mesg (str): The message to log.

extra (dict): Extra key / value pairs to include when structured logging is enabled on the Cortex.

Returns:
The return type is `null`.

13.1.49 $lib.macro

A Storm Library for interacting with the Storm Macros in the Cortex.

$lib.macro.del(name)

Delete a Storm Macro by name from the Cortex.

Args:
name (str): The name of the macro to delete.

Returns:
The return type is `null`.

$lib.macro.get(name)

Get a Storm Macro definition by name from the Cortex.

Args:
name (str): The name of the macro to get.

Returns:
A macro definition. The return type is *dict*.

$lib.macro.grant(name, scope, iden, level)

Modify permissions granted to users/roles on a Storm Macro.

Args:
name (str): Name of the Storm Macro to modify.

scope (str): The scope, either "users" or "roles".

iden (str): The user/role iden depending on scope.

level (int): The permission level number.

Returns:
The return type is `null`.

$lib.macro.list()

Get a list of Storm Macros in the Cortex.

Returns:
A list of `dict` objects containing Macro definitions. The return type is *list*.

$lib.macro.mod(name, info)

Modify user editable properties of a Storm Macro.

Args:
name (str): Name of the Storm Macro to modify.

info (dict): A dictionary of the properties to edit.

Returns:
The return type is `null`.

$lib.macro.set(name, storm)

Add or modify an existing Storm Macro in the Cortex.

Args:
name (str): Name of the Storm Macro to add or modify.

storm: The Storm query to add to the macro. The input type may be one of the following: `str`, `storm:query`.

Returns:
The return type is `null`.

13.1.50 $lib.math

A Storm library for performing math operations.

$lib.math.number(value)

Convert a value to a Storm Number object.

Storm Numbers are high precision fixed point decimals corresponding to the hugenum storage type.

This is not to be used for converting a string to an integer.

Args:
value (any): Value to convert.

Returns:
A Number object. The return type is *number*.

13.1.51 $lib.mime.html

A Storm library for manipulating HTML text.

$lib.mime.html.totext(html)

Return inner text from all tags within an HTML document.

Args:
html (str): The HTML text to be parsed.

Returns:
The newline-joined inner HTML text. The return type is *str*.

13.1.52 $lib.model

A Storm Library for interacting with the Data Model in the Cortex.

$lib.model.form(name)

Get a form object by name.

Args:
name (str): The name of the form to retrieve.

Returns:
The `model:form` instance if the form is present or null. The return type may be one of the following: *model:form*, `null`.

$lib.model.prop(name)

Get a prop object by name.

Args:
name (str): The name of the prop to retrieve.

Returns:
The `model:property` instance if the type if present or null. The return type may be one of the following: *model:property*, `null`.

$lib.model.tagprop(name)

Get a tag property object by name.

Args:
name (str): The name of the tag prop to retrieve.

Returns:
The `model:tagprop` instance if the tag prop if present or null. The return type may be one of the following: *model:tagprop*, `null`.

$lib.model.type(name)

Get a type object by name.

Args:
name (str): The name of the type to retrieve.

Returns:
The `model:type` instance if the type if present on the form or null. The return type may be one of the following: *model:type*, `null`.

13.1.53 $lib.model.deprecated

A storm library for interacting with the model deprecation mechanism.

$lib.model.deprecated.lock(name, locked)

Set the locked property for a deprecated model element.

Args:
name (str): The full path of the model element to lock.

locked (boolean): The lock status.

Returns:
The return type is `null`.

$lib.model.deprecated.locks()

Get a dictionary of the data model elements which are deprecated and their lock status in the Cortex.

Returns:
A dictionary of named elements to their boolean lock values. The return type is *dict*.

13.1.54 $lib.model.edge

A Storm Library for interacting with light edges and manipulating their key-value attributes.

$lib.model.edge.del(verb, key)

Delete a key from the key-value store for a verb.

Args:
verb (str): The name of the Edge verb to remove a key from.

key (str): The name of the key to remove from the key-value store.

Returns:
The return type is `null`.

$lib.model.edge.get(verb)

Get the key-value data for a given Edge verb.

Args:
verb (str): The Edge verb to look up.

Returns:
A dictionary representing the key-value data set on a verb. The return type is *dict*.

$lib.model.edge.list()

Get a list of (verb, key-value dictionary) pairs for Edge verbs in the current Cortex View.

Returns:
A list of (str, dict) tuples for each verb in the current Cortex View. The return type is *list*.

$lib.model.edge.set(verb, key, valu)

Set a key-value for a given Edge verb.

Args:
verb (str): The Edge verb to set a value for.

key (str): The key to set.

valu (str): The value to set.

Returns:
The return type is `null`.

$lib.model.edge.validkeys()

Get a list of the valid keys that can be set on an Edge verb.

Returns:
A list of the valid keys. The return type is *list*.

13.1.55 $lib.model.ext

A Storm library for manipulating extended model elements.

$lib.model.ext.addExtModel(model)

Add extended model elements to the Cortex from getExtModel().

Args:
model (dict): A model dictionary from getExtModel().

Returns:
The return type is *boolean*.

$lib.model.ext.addForm(formname, basetype, typeopts, typeinfo)

Add an extended form definition to the data model.

Args:
formname (str): The name of the form to add.

basetype (str): The base type the form is derived from.

typeopts (dict): A Synapse type opts dictionary.

typeinfo (dict): A Synapse form info dictionary.

Returns:
The return type is `null`.

$lib.model.ext.addFormProp(formname, propname, typedef, propinfo)

Add an extended property definition to the data model.

Args:
formname (str): The name of the form to add the property to.

propname (str): The name of the extended property.

typedef (list): A Synapse type definition tuple.

propinfo (dict): A synapse property definition dictionary.

Returns:
The return type is `null`.

$lib.model.ext.addTagProp(propname, typedef, propinfo)

Add an extended tag property definition to the data model.

Args:

propname (str): The name of the tag property.

typedef (list): A Synapse type definition tuple.

propinfo (dict): A synapse property definition dictionary.

Returns:

The return type is `null`.

$lib.model.ext.addUnivProp(propname, typedef, propinfo)

Add an extended universal property definition to the data model.

Args:

propname (str): The name of the universal property.

typedef (list): A Synapse type definition tuple.

propinfo (dict): A synapse property definition dictionary.

Returns:

The return type is `null`.

$lib.model.ext.delForm(formname)

Remove an extended form definition from the model.

Args:

formname (str): The extended form to remove.

Returns:

The return type is `null`.

$lib.model.ext.delFormProp(formname, propname)

Remove an extended property definition from the model.

Args:

formname (str): The form with the extended property.

propname (str): The extended property to remove.

Returns:

The return type is `null`.

$lib.model.ext.delTagProp(propname)

Remove an extended tag property definition from the model.

Args:
propname (str): Name of the tag property to remove.

Returns:
The return type is `null`.

$lib.model.ext.delUnivProp(propname)

Remove an extended universal property definition from the model.

Args:
propname (str): Name of the universal property to remove.

Returns:
The return type is `null`.

$lib.model.ext.getExtModel()

Get all extended model elements.

Returns:
The return type is *dict*.

13.1.56 $lib.model.tags

A Storm Library for interacting with tag specifications in the Cortex Data Model.

$lib.model.tags.del(tagname)

Delete a tag model specification.

Examples:
Delete the tag model specification for `cno.threat`:

```
$lib.model.tags.del(cno.threat)
```

Args:
tagname (str): The name of the tag.

Returns:
The return type is `null`.

$lib.model.tags.get(tagname)

Retrieve a tag model specification.

Examples:

Get the tag model specification for `cno.threat`:

```
$dict = $lib.model.tags.get(cno.threat)
```

Args:

tagname (str): The name of the tag.

Returns:

The tag model definition. The return type is *dict*.

$lib.model.tags.list()

List all tag model specifications.

Examples:

Iterate over the tag model specifications in the Cortex:

```
for ($name, $info) in $lib.model.tags.list() {
    ...
}
```

Returns:

List of tuples containing the tag name and model definition The return type is *list*.

$lib.model.tags.pop(tagname, propname)

Pop and return a tag model property.

Examples:

Remove the regex list from the `cno.threat` tag model:

```
$regxlist = $lib.model.tags.pop(cno.threat, regex)
```

Args:

tagname (str): The name of the tag.

propname (str): The name of the tag model property.

Returns:

The value of the property. The return type is `prim`.

$lib.model.tags.set(tagname, propname, propvalu)

Set a tag model property for a tag.

Examples:
Create a tag model for the `cno.cve` tag:

```
$regx = ($lib.null, $lib.null, "[0-9]{4}", "[0-9]{5}")
$lib.model.tags.set(cno.cve, regex, $regx)
```

Args:
tagname (str): The name of the tag.

propname (str): The name of the tag model property.

propvalu (prim): The value to set.

Returns:
The return type is `null`.

13.1.57 $lib.notifications

A Storm library for a user interacting with their notifications.

$lib.notifications.del(indx)

Delete a previously delivered notification.

Args:
indx (int): The index number of the notification to delete.

Retn:
Returns an ($ok, $valu) tuple. The return type is *list*.

$lib.notifications.get(indx)

Return a notification by ID (or $lib.null).

Args:
indx (int): The index number of the notification to return.

Retn:
The requested notification or $lib.null. The return type is *dict*.

$lib.notifications.list(size=$lib.null)

Yield (<indx>, <mesg>) tuples for a user's notifications.

Args:
size (int): The max number of notifications to yield.

Yields:
Yields (useriden, time, mesgtype, msgdata) tuples. The return type is *list*.

13.1.58 $lib.pack

Packing / unpacking structured bytes.

$lib.pack.en(fmt, items)

Pack a sequence of items into an array of bytes.

Args:

fmt (str): A python struct.pack() format string.

items (list): A list of values to be packed.

Returns:

The packed byte structure. The return type is *bytes*.

$lib.pack.un(fmt, byts, offs=(0))

Unpack a sequence of items from an array of bytes.

Args:

fmt (str): A python struct.unpack() format string.

byts (bytes): Bytes to be unpacked

offs (int): The offset to begin unpacking from.

Returns:

The unpacked items. The return type is *list*.

13.1.59 $lib.pipe

A Storm library for interacting with non-persistent queues.

$lib.pipe.gen(filler, size=(10000))

Generate and return a Storm Pipe.

Notes:

The filler query is run in parallel with $pipe. This requires the permission `storm.pipe.gen` to use.

Examples:

Fill a pipe with a query and consume it with another:

```
$pipe = $lib.pipe.gen(${ $pipe.puts((1, 2, 3)) })

for $items in $pipe.slices(size=2) {
    $dostuff($items)
}
```

Args:

filler: A Storm query to fill the Pipe. The input type may be one of the following: `str`, `storm:query`.

size (int): Maximum size of the pipe.

Returns:

The pipe containing query results. The return type is *pipe*.

13.1.60 $lib.pkg

A Storm Library for interacting with Storm Packages.

$lib.pkg.add(pkgdef, verify=$lib.false)

Add a Storm Package to the Cortex.

Args:
  pkgdef (dict): A Storm Package definition.

  verify (boolean): Verify storm package signature.

Returns:
  The return type is `null`.

$lib.pkg.del(name)

Delete a Storm Package from the Cortex.

Args:
  name (str): The name of the package to delete.

Returns:
  The return type is `null`.

$lib.pkg.deps(pkgdef)

Verify the dependencies for a Storm Package.

Args:
  pkgdef (dict): A Storm Package definition.

Returns:
  A dictionary listing dependencies and if they are met. The return type is *dict*.

$lib.pkg.get(name)

Get a Storm Package from the Cortex.

Args:
  name (str): A Storm Package name.

Returns:
  The Storm package definition. The return type is *dict*.

$lib.pkg.has(name)

Check if a Storm Package is available in the Cortex.

Args:
name (str): A Storm Package name to check for the existence of.

Returns:
True if the package exists in the Cortex, False if it does not. The return type is *boolean*.

$lib.pkg.list()

Get a list of Storm Packages loaded in the Cortex.

Returns:
A list of Storm Package definitions. The return type is *list*.

13.1.61 $lib.projects

A Storm Library for interacting with Projects in the Cortex.

$lib.projects.add(name, desc=)

Add a new project

Args:
name (str): The name of the Project to add

desc (str): A description of the overall project

Returns:
The newly created *proj:project* object The return type is *proj:project*.

$lib.projects.del(name)

Delete an existing project

Args:
name (str): The name of the Project to delete

Returns:
True if the project exists and gets deleted, otherwise False The return type is *boolean*.

$lib.projects.get(name)

Retrieve a project by name

Args:
name (str): The name of the Project to get

Returns:
The *proj:project `object, if it exists, otherwise null The return type is :ref:`stormprims-proj-project-f527*.

13.1.62 $lib.ps

A Storm Library for interacting with running tasks on the Cortex.

$lib.ps.kill(prefix)

Stop a running task on the Cortex.

Args:
prefix (str): The prefix of the task to stop. Tasks will only be stopped if there is a single prefix match.

Returns:
True if the task was cancelled, False otherwise. The return type is *boolean*.

$lib.ps.list()

List tasks the current user can access.

Returns:
A list of task definitions. The return type is *list*.

13.1.63 $lib.queue

A Storm Library for interacting with persistent Queues in the Cortex.

$lib.queue.add(name)

Add a Queue to the Cortex with a given name.

Args:
name (str): The name of the queue to add.

Returns:
The return type is *queue*.

$lib.queue.del(name)

Delete a given named Queue.

Args:
name (str): The name of the queue to delete.

Returns:
The return type is `null`.

$lib.queue.gen(name)

Add or get a Storm Queue in a single operation.

Args:
name (str): The name of the Queue to add or get.

Returns:
The return type is *queue*.

$lib.queue.get(name)

Get an existing Storm Queue object.

Args:
name (str): The name of the Queue to get.

Returns:
A `queue` object. The return type is *queue*.

$lib.queue.list()

Get a list of the Queues in the Cortex.

Returns:
A list of queue definitions the current user is allowed to interact with. The return type is *list*.

13.1.64 $lib.random

A Storm library for generating random values.

$lib.random.int(maxval, minval=(0))

Generate a random integer.

Args:
maxval (int): The maximum random value.

minval (int): The minimum random value.

Returns:
A random integer in the range min-max inclusive. The return type is `int`.

13.1.65 $lib.regex

A Storm library for searching/matching with regular expressions.

$lib.regex.escape(text)

Escape arbitrary strings for use in a regular expression pattern.

Example:

Escape node values for use in a regex pattern:

```
for $match in $lib.regex.findall($lib.regex.escape($node.repr()), $mydocument)
↪{
    // do something with $match
}
```

Escape node values for use in regular expression filters:

```
it:dev:str~=$lib.regex.escape($node.repr())
```

Args:
text (str): The text to escape.

Returns:
Input string with special characters escaped. The return type is *str*.

$lib.regex.findall(pattern, text, flags=(0))

Search the given text for the patterns and return a list of matching strings.

Note:
If multiple matching groups are specified, the return value is a list of lists of strings.

Example:

Extract the matching strings from a piece of text:

```
for $x in $lib.regex.findall("G[0-9]{4}", "G0006 and G0001") {
    $dostuff($x)
}
```

Args:
pattern (str): The regular expression pattern.

text (str): The text to match.

flags (int): Regex flags to control the match behavior.

Returns:
A list of lists of strings for the matching groups in the pattern. The return type is *list*.

$lib.regex.flags.i

Regex flag to indicate that case insensitive matches are allowed.

Returns:
The type is `int`.

$lib.regex.flags.m

Regex flag to indicate that multiline matches are allowed.

Returns:
The type is `int`.

$lib.regex.matches(pattern, text, flags=(0))

Check if text matches a pattern. Returns $lib.true if the text matches the pattern, otherwise $lib.false.

Notes:
This API requires the pattern to match at the start of the string.

Example:
Check if the variable matches a expression:

```
if $lib.regex.matches("^[0-9]+.[0-9]+.[0-9]+$", $text) {
    $lib.print("It's semver! ...probably")
}
```

Args:
pattern (str): The regular expression pattern.

text (str): The text to match.

flags (int): Regex flags to control the match behavior.

Returns:
True if there is a match, False otherwise. The return type is *boolean*.

$lib.regex.replace(pattern, replace, text, flags=(0))

Replace any substrings that match the given regular expression with the specified replacement.

Example:
Replace a portion of a string with a new part based on a regex:

```
$norm = $lib.regex.replace("\sAND\s", " & ", "Ham and eggs!", $lib.regex.flags.i)
```

Args:
pattern (str): The regular expression pattern.

replace (str): The text to replace matching sub strings.

text (str): The input text to search/replace.

flags (int): Regex flags to control the match behavior.

Returns:
The new string with matches replaced. The return type is *str*.

$lib.regex.search(pattern, text, flags=(0))

Search the given text for the pattern and return the matching groups.

Note:
In order to get the matching groups, patterns must use parentheses to indicate the start and stop of the regex to return portions of. If groups are not used, a successful match will return a empty list and a unsuccessful match will return `$lib.null`.

Example:
Extract the matching groups from a piece of text:

```
$m = $lib.regex.search("^([0-9])+.([0-9])+.([0-9])+$", $text)
if $m {
    ($maj, $min, $pat) = $m
}
```

Args:
pattern (str): The regular expression pattern.

text (str): The text to match.

flags (int): Regex flags to control the match behavior.

Returns:
A list of strings for the matching groups in the pattern. The return type is *list*.

13.1.66 $lib.scrape

A Storm Library for providing helpers for scraping nodes from text.

$lib.scrape.context(text)

Attempt to scrape information from a blob of text, getting the context information about the values found.

Notes:
This does call the `scrape` Storm interface if that behavior is enabled on the Cortex.

Examples:
Scrape some text and make nodes out of it:

```
for ($form, $valu, $info) in $lib.scrape.context($text) {
    [ ( *$form ?= $valu ) ]
}
```

Args:
text (str): The text to scrape

Yields:
A dictionary of scraped values, rule types, and offsets scraped from the text. The return type is *dict*.

$lib.scrape.genMatches(text, pattern, fangs=$lib.null, flags=(2))

genMatches is a generic helper function for constructing scrape interfaces using pure Storm.

It accepts the text, a regex pattern, and produce results that can easily be used to create

Notes:
The pattern must have a named regular expression match for the key `valu` using the named group syntax. For example `(somekey\s)(?P<valu>[a-z0-9]+)\s`.

Examples:
A scrape implementation with a regex that matches name keys in text:

```
$re="(Name\:\s)(?P<valu>[a-z0-9]+)\s"
$form="ps:name"

function scrape(text, form) {
        $ret = $lib.list()
        for ($valu, $info) in $lib.scrape.genMatches($text, $re) {
            $ret.append(($form, $valu, $info))
        }
        return ( $ret )
    }
```

Args:
text (str): The text to scrape

pattern (str): The regular expression pattern to match against.

fangs (list): A list of (src, dst) pairs to refang from text. The src must be equal or larger than the dst in length.

flags (int): Regex flags to use (defaults to IGNORECASE).

Yields:
The return type is *list*.

$lib.scrape.ndefs(text)

Attempt to scrape node form, value tuples from a blob of text.

Examples:
Scrape some text and attempt to make nodes out of it:

```
for ($form, $valu) in $lib.scrape($text) {
    [ ( *$form ?= $valu ) ]
}
```

Args:
text (str): The text to scrape

Yields:
A list of (form, value) tuples scraped from the text. The return type is *list*.

13.1.67 $lib.service

A Storm Library for interacting with Storm Services.

$lib.service.add(name, url)

Add a Storm Service to the Cortex.

Args:
name (str): Name of the Storm Service to add.

url (str): The Telepath URL to the Storm Service.

Returns:
The Storm Service definition. The return type is *dict*.

$lib.service.del(iden)

Remove a Storm Service from the Cortex.

Args:
iden (str): The iden of the service to remove.

Returns:
The return type is `null`.

$lib.service.get(name)

Get a Storm Service definition.

Args:
name (str): The local name, local iden, or remote name, of the service to get the definition for.

Returns:
A Storm Service definition. The return type is *dict*.

$lib.service.has(name)

Check if a Storm Service is available in the Cortex.

Args:
name (str): The local name, local iden, or remote name, of the service to check for the existence of.

Returns:
True if the service exists in the Cortex, False if it does not. The return type is *boolean*.

$lib.service.list()

List the Storm Service definitions for the Cortex.

Notes:
The definition dictionaries have an additional `ready` key added to them to indicate if the Cortex is currently connected to the Storm Service or not.

Returns:
A list of Storm Service definitions. The return type is *list*.

$lib.service.wait(name, timeout=$lib.null)

Wait for a given service to be ready.

Notes:
If a timeout value is not specified, this will block a Storm query until the service is available.

Args:
name (str): The name, or iden, of the service to wait for.

timeout (int): Number of seconds to wait for the service.

Returns:
Returns true if the service is available, false on a timeout waiting for the service to be ready. The return type is *boolean*.

13.1.68 $lib.spooled

A Storm Library for interacting with Spooled Objects.

$lib.spooled.set(*vals)

Get a Spooled Storm Set object.

A Spooled Storm Set object is memory-safe to grow to extraordinarily large sizes, as it will fallback to file backed storage, with two restrictions. First is that all items in the set can be serialized to a file if the set grows too large, so all items added must be a serializable Storm primitive. Second is that when an item is added to the Set, because it could be immediately written disk, do not hold any references to it outside of the Set itself, as the two objects could differ.

Args:
*vals (any): Initial values to place in the set.

Returns:
The new set. The return type is *set*.

13.1.69 $lib.stats

A Storm Library for statistics related functionality.

$lib.stats.tally()

Get a Tally object.

Returns:
A new tally object. The return type is *stat:tally*.

13.1.70 $lib.stix

A Storm Library for interacting with Stix Version 2.1 CS02.

$lib.stix.lift(bundle)

Lift nodes from a STIX Bundle made by Synapse.

Notes:
This lifts nodes using the Node definitions embedded into the bundle when created by Synapse using custom extension properties.

Examples:
Lifting nodes from a STIX bundle:

```
yield $lib.stix($bundle)
```

Args:
bundle (dict): The STIX bundle to lift nodes from.

Yields:
Yields nodes The return type is *node*.

$lib.stix.validate(bundle)

Validate a STIX Bundle.

Notes:
This returns a dictionary containing the following values:

```
{
    'ok': <boolean> - False if bundle is invalid, True otherwise.
    'mesg': <str> - An error message if there was an error when validating the␣
↪bundle.
    'results': The results of validating the bundle.
}
```

Args:
bundle (dict): The stix bundle to validate.

Returns:
Results dictionary. The return type is *dict*.

13.1.71 $lib.stix.export

A Storm Library for exporting to STIX version 2.1 CS02.

$lib.stix.export.bundle(config=$lib.null)

Return a new empty STIX bundle.

The config argument maps synapse forms to stix types and allows you to specify how to resolve STIX properties and relationships. The config expects to following format:

```
{
    "maxsize": 10000,

    "forms": {
        <formname>: {
            "default": <stixtype0>,
            "stix": {
                <stixtype0>: {
                    "props": {
                        <stix_prop_name>: <storm_with_return>,
                        ...
                    },
                    "rels": (
                        ( <relname>, <target_stixtype>, <storm> ),
                        ...
                    ),
                    "revs": (
                        ( <revname>, <source_stixtype>, <storm> ),
                        ...
                    )
                },
                <stixtype1>: ...
            },
        },
    },
},
```

For example, the default config includes the following entry to map ou:campaign nodes to stix campaigns:

```
{ "forms": {
    "ou:campaign": {
        "default": "campaign",
        "stix": {
            "campaign": {
                "props": {
                    "name": "{+:name return(:name)} return($node.repr())",
                    "description": "+:desc return(:desc)",
                    "objective": "+:goal :goal -> ou:goal +:name return(:name)",
                    "created": "return($lib.stix.export.timestamp(.created))",
                    "modified": "return($lib.stix.export.timestamp(.created))",
                },
                "rels": (
```

(continues on next page)

(continued from previous page)

```
                    ("attributed-to", "threat-actor", ":org -> ou:org"),
                    ("originates-from", "location", ":org -> ou:org :hq -> geo:place"),
                    ("targets", "identity", "-> risk:attack :target:org -> ou:org"),
                    ("targets", "identity", "-> risk:attack :target:person -> ps:person
↪"),
                ),
            },
        },
}},
```

You may also specify pivots on a per form+stixtype basis to automate pivoting to additional nodes to include in the bundle:

```
{"forms": {
    "inet:fqdn":
        ...
        "domain-name": {
            ...
            "pivots": [
                {"storm": "-> inet:dns:a -> inet:ipv4", "stixtype": "ipv4-addr"}
            ]
        {
    }
}
```

Note:
The default config is an evolving set of mappings. If you need to guarantee stable output please specify a config.

Args:
config (dict): The STIX bundle export config to use.

Returns:
A new `stix:bundle` instance. The return type is *stix:bundle*.

$lib.stix.export.config()

Construct a default STIX bundle export config.

Returns:
A default STIX bundle export config. The return type is *dict*.

$lib.stix.export.timestamp(tick)

Format an epoch milliseconds timestamp for use in STIX output.

Args:
tick (time): The epoch milliseconds timestamp.

Returns:
A STIX formatted timestamp string. The return type is *str*.

13.1.72 $lib.stix.import

A Storm Library for importing Stix Version 2.1 data.

$lib.stix.import.config()

Return an editable copy of the default STIX ingest config.

Returns:
A copy of the default STIX ingest configuration. The return type is *dict*.

$lib.stix.import.ingest(bundle, config=$lib.null)

Import nodes from a STIX bundle.

Args:
bundle (dict): The STIX bundle to ingest.

config (dict): An optional STIX ingest configuration.

Yields:
Yields nodes The return type is *node*.

13.1.73 $lib.storm

A Storm library for evaluating dynamic storm expressions.

$lib.storm.eval(text, cast=$lib.null)

Evaluate a storm runtime value and optionally cast/coerce it.

NOTE: If storm logging is enabled, the expression being evaluated will be logged separately.

Args:
text (str): A storm expression string.

cast (str): A type to cast the result to.

Returns:
The value of the expression and optional cast. The return type is `any`.

13.1.74 $lib.str

A Storm Library for interacting with strings.

$lib.str.concat(*args)

Concatenate a set of strings together.

Args:
*args (any): Items to join together.

Returns:
The joined string. The return type is *str*.

$lib.str.format(text, **kwargs)

Format a text string.

Examples:
Format a string with a fixed argument and a variable:

```
cli> storm $list=(1,2,3,4)
     $str=$lib.str.format('Hello {name}, your list is {list}!', name='Reader', list=
↪$list)
     $lib.print($str)

Hello Reader, your list is ['1', '2', '3', '4']!
```

Args:
text (str): The base text string.

**kwargs (any): Keyword values which are substituted into the string.

Returns:
The new string. The return type is *str*.

$lib.str.join(sepr, items)

Join items into a string using a separator.

Examples:
Join together a list of strings with a dot separator:

```
cli> storm $foo=$lib.str.join('.', ('rep', 'vtx', 'tag')) $lib.print($foo)

rep.vtx.tag
```

Args:
sepr (str): The separator used to join strings with.

items (list): A list of items to join together.

Returns:
The joined string. The return type is *str*.

13.1.75 $lib.tags

Storm utility functions for tags.

$lib.tags.prefix(names, prefix, ispart=$lib.false)

Normalize and prefix a list of syn:tag:part values so they can be applied.

Examples:
Add tag prefixes and then use them to tag nodes:

```
$tags = $lib.tags.prefix($result.tags, vtx.visi)
{ for $tag in $tags { [ +#$tag ] } }
```

Args:
names (list): A list of syn:tag:part values to normalize and prefix.

prefix (str): The string prefix to add to the syn:tag:part values.

ispart (boolean): Whether the names have already been normalized. Normalization will be skipped if set to true.

Returns:
A list of normalized and prefixed syn:tag values. The return type is *list*.

13.1.76 $lib.telepath

A Storm Library for making Telepath connections to remote services.

$lib.telepath.open(url)

Open and return a Telepath RPC proxy.

Args:
url (str): The Telepath URL to connect to.

Returns:
A object representing a Telepath Proxy. The return type is *telepath:proxy*.

13.1.77 $lib.time

A Storm Library for interacting with timestamps.

$lib.time.day(tick)

Returns the day part of a time value.

Args:
tick (time): A time value.

Returns:
The day part of the time expression. The return type is `int`.

$lib.time.dayofmonth(tick)

Returns the index (beginning with 0) of the day within the month.

Args:
tick (time): A time value.

Returns:
The index of the day within month. The return type is `int`.

$lib.time.dayofweek(tick)

Returns the index (beginning with monday as 0) of the day within the week.

Args:
tick (time): A time value.

Returns:
The index of the day within week. The return type is `int`.

$lib.time.dayofyear(tick)

Returns the index (beginning with 0) of the day within the year.

Args:
tick (time): A time value.

Returns:
The index of the day within year. The return type is `int`.

$lib.time.format(valu, format)

Format a Synapse timestamp into a string value using `datetime.strftime()`.

Examples:
Format a timestamp into a string:

```
cli> storm $now=$lib.time.now() $str=$lib.time.format($now, '%A %d, %B %Y') $lib.
↪print($str)

Tuesday 14, July 2020
```

Args:
valu (int): A timestamp in epoch milliseconds.

format (str): The strftime format string.

Returns:
The formatted time string. The return type is *str*.

$lib.time.fromunix(secs)

Normalize a timestamp from a unix epoch time in seconds to milliseconds.

Examples:
Convert a timestamp from seconds to millis and format it:

```
cli> storm $seconds=1594684800 $millis=$lib.time.fromunix($seconds)
     $str=$lib.time.format($millis, '%A %d, %B %Y') $lib.print($str)

Tuesday 14, July 2020
```

Args:
secs (int): Unix epoch time in seconds.

Returns:
The normalized time in milliseconds. The return type is `int`.

$lib.time.hour(tick)

Returns the hour part of a time value.

Args:
tick (time): A time value.

Returns:
The hour part of the time expression. The return type is `int`.

$lib.time.minute(tick)

Returns the minute part of a time value.

Args:
tick (time): A time value.

Returns:
The minute part of the time expression. The return type is `int`.

$lib.time.month(tick)

Returns the month part of a time value.

Args:
tick (time): A time value.

Returns:
The month part of the time expression. The return type is `int`.

$lib.time.monthofyear(tick)

Returns the index (beginning with 0) of the month within the year.

Args:
tick (time): A time value.

Returns:
The index of the month within year. The return type is `int`.

$lib.time.now()

Get the current epoch time in milliseconds.

Returns:
Epoch time in milliseconds. The return type is `int`.

$lib.time.parse(valu, format, errok=$lib.false)

Parse a timestamp string using `datetime.strptime()` into an epoch timestamp.

Examples:
Parse a string as for its month/day/year value into a timestamp:

```
cli> storm $s='06/01/2020' $ts=$lib.time.parse($s, '%m/%d/%Y') $lib.print($ts)

1590969600000
```

Args:
valu (str): The timestamp string to parse.

format (str): The format string to use for parsing.

errok (boolean): If set, parsing errors will return `$lib.null` instead of raising an exception.

Returns:
The epoch timestamp for the string. The return type is `int`.

$lib.time.second(tick)

Returns the second part of a time value.

Args:
tick (time): A time value.

Returns:
The second part of the time expression. The return type is `int`.

$lib.time.sleep(valu)

Pause the processing of data in the storm query.

Notes:
This has the effect of clearing the Snap's cache, so any node lifts performed after the `$lib.time.sleep(...)` executes will be lifted directly from storage.

Args:
valu (int): The number of seconds to pause for.

Returns:
The return type is `null`.

$lib.time.ticker(tick, count=$lib.null)

Periodically pause the processing of data in the storm query.

Notes:
This has the effect of clearing the Snap's cache, so any node lifts performed after each tick will be lifted directly from storage.

Args:
tick (int): The amount of time to wait between each tick, in seconds.

count (int): The number of times to pause the query before exiting the loop. This defaults to None and will yield forever if not set.

Yields:
This yields the current tick count after each time it wakes up. The return type is `int`.

$lib.time.toUTC(tick, timezone)

Adjust an epoch milliseconds timestamp to UTC from the given timezone.

Args:
tick (time): A time value.

timezone (str): A timezone name. See python pytz docs for options.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

$lib.time.year(tick)

Returns the year part of a time value.

Args:
tick (time): A time value.

Returns:
The year part of the time expression. The return type is `int`.

13.1.78 $lib.trigger

A Storm Library for interacting with Triggers in the Cortex.

$lib.trigger.add(tdef)

Add a Trigger to the Cortex.

Args:
tdef (dict): A Trigger definition.

Returns:
The new trigger. The return type is *trigger*.

$lib.trigger.del(prefix)

Delete a Trigger from the Cortex.

Args:
prefix (str): A prefix to match in order to identify a trigger to delete. Only a single matching prefix will be deleted.

Returns:
The iden of the deleted trigger which matched the prefix. The return type is *str*.

$lib.trigger.disable(prefix)

Disable a Trigger in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a trigger to disable. Only a single matching prefix will be disabled.

Returns:
The iden of the trigger that was disabled. The return type is *str*.

$lib.trigger.enable(prefix)

Enable a Trigger in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a trigger to enable. Only a single matching prefix will be enabled.

Returns:
The iden of the trigger that was enabled. The return type is *str*.

$lib.trigger.get(iden)

Get a Trigger in the Cortex.

Args:
iden (str): The iden of the Trigger to get.

Returns:
The requested `trigger` object. The return type is *trigger*.

$lib.trigger.list(all=$lib.false)

Get a list of Triggers in the current view or every view.

Args:
all (boolean): Get a list of all the readable Triggers in every readable View.

Returns:
A list of `trigger` objects the user is allowed to access. The return type is *list*.

$lib.trigger.mod(prefix, query)

Modify an existing Trigger in the Cortex.

Args:
prefix (str): A prefix to match in order to identify a trigger to modify. Only a single matching prefix will be modified.

query: The new Storm query to set as the trigger query. The input type may be one of the following: `str`, `storm:query`.

Returns:
The iden of the modified Trigger The return type is *str*.

13.1.79 $lib.user

A Storm Library for interacting with data about the current user.

$lib.user.allowed(permname, gateiden=$lib.null, default=$lib.false)

Check if the current user has a given permission.

Args:
permname (str): The permission string to check.

gateiden (str): The authgate iden.

default (boolean): The default value.

Returns:
True if the user has the requested permission, false otherwise. The return type is *boolean*.

$lib.user.iden

The user GUID for the current storm user.

Returns:
The type is *str*.

$lib.user.name()

Get the name of the current runtime user.

Returns:
The username. The return type is *str*.

$lib.user.profile

Get a Hive dictionary representing the current user's profile information.

Returns:
The type is *hive:dict*.

$lib.user.vars

Get a Hive dictionary representing the current user's persistent variables.

Returns:
The type is *hive:dict*.

13.1.80 $lib.vars

A Storm Library for interacting with runtime variables.

$lib.vars.del(name)

Unset a variable in the current Runtime.

Args:
name (str): The variable name to remove.

Returns:
The return type is `null`.

$lib.vars.get(name, defv=$lib.null)

Get the value of a variable from the current Runtime.

Args:
name (str): Name of the variable to get.

defv (prim): The default value returned if the variable is not set in the runtime.

Returns:
The value of the variable. The return type is `any`.

$lib.vars.list()

Get a list of variables from the current Runtime.

Returns:
A list of variable names and their values for the current Runtime. The return type is *list*.

$lib.vars.set(name, valu)

Set the value of a variable in the current Runtime.

Args:
name (str): Name of the variable to set.

valu (prim): The value to set the variable too.

Returns:
The return type is `null`.

$lib.vars.type(valu)

Get the type of the argument value.

Args:
valu (any): Value to inspect.

Returns:
The type of the argument. The return type is *str*.

13.1.81 $lib.vault

A Storm Library for interacting with vaults.

$lib.vault.add(name, vtype, scope, owner, secrets, configs)

Create a new vault.

Args:
name (str): The name of the new vault.

vtype (str): The type of this vault.

scope (str): Scope for this vault. One of "user", "role", "global", or $lib.null for unscoped vaults.

owner (str): User/role iden for this vault if scope is "user" or "role". None for "global" scope vaults.

secrets (dict): The initial secret data to store in this vault.

configs (dict): The initial config data to store in this vault.

Returns:
Iden of the newly created vault. The return type is *str*.

$lib.vault.byname(name)

Get a vault by name.

Args:

name (str):
The name of the vault to retrieve. If user only has PERM_READ, the secrets data will not be returned. If the user has PERM_EDIT or higher, secrets data will be included in the vault.

Returns:
The requested vault. The return type is *dict*.

$lib.vault.bytype(vtype, scope=$lib.null)

Get a vault for a specified vault type.

Args:
vtype (str): The vault type to retrieved.

scope (str): The scope for the specified type. If $lib.null, then getByType will search.

Returns:
Vault or $lib.null if the vault could not be retrieved. The return type is *vault*.

$lib.vault.get(iden)

Get a vault by iden.

Args:

iden (str):
The iden of the vault to retrieve. If user only has PERM_READ, the secrets data will not be returned. If the user has PERM_EDIT or higher, secrets data will be included in the vault.

Returns:
The requested vault. The return type is *dict*.

$lib.vault.list()

List vaults accessible to the current user.

Returns:
Yields vaults. The return type is *list*.

$lib.vault.print(vault)

Print the details of the specified vault.

Args:
vault (dict): The vault to print.

Returns:
The return type is `null`.

13.1.82 $lib.version

A Storm Library for interacting with version information.

$lib.version.commit()

The synapse commit hash for the local Cortex.

Returns:
The commit hash. The return type is *str*.

$lib.version.matches(vertup, reqstr)

Check if the given version triple meets the requirements string.

Examples:
Check if the synapse version is in a range:

```
$synver = $lib.version.synapse()
if $lib.version.matches($synver, ">=2.9.0") {
    $dostuff()
}
```

Args:
vertup (list): Triple of major, minor, and patch version integers.

reqstr (str): The version string to compare against.

Returns:
True if the version meets the requirements, False otherwise. The return type is *boolean*.

$lib.version.synapse()

The synapse version tuple for the local Cortex.

Returns:
The version triple. The return type is *list*.

13.1.83 $lib.view

A Storm Library for interacting with Views in the Cortex.

$lib.view.add(layers, name=$lib.null, worldreadable=$lib.false)

Add a View to the Cortex.

Args:
layers (list): A list of layer idens which make up the view.

name (str): The name of the view.

worldreadable (boolean): Grant read access to the *all* role.

Returns:
A `view` object representing the new View. The return type is *view*.

$lib.view.del(iden)

Delete a View from the Cortex.

Args:
iden (str): The iden of the View to delete.

Returns:
The return type is `null`.

$lib.view.get(iden=$lib.null)

Get a View from the Cortex.

Args:
iden (str): The iden of the View to get. If not specified, returns the current View.

Returns:
The storm view object. The return type is *view*.

$lib.view.list(deporder=$lib.false)

List the Views in the Cortex.

Args:
deporder (bool): Return the lists in bottom-up dependency order.

Returns:
List of `view` objects. The return type is *list*.

13.1.84 $lib.xml

A Storm library for parsing XML.

$lib.xml.parse(valu)

Parse an XML string into an xml:element tree.

Args:
valu (str): The XML string to parse into an xml:element tree.

Returns:
An xml:element for the root node of the XML tree. The return type is *xml:element*.

13.1.85 $lib.yaml

A Storm Library for saving/loading YAML data.

$lib.yaml.load(valu)

Decode a YAML string/bytes into an object.

Args:
valu (str): The string to decode.

Returns:
The decoded primitive object. The return type is `prim`.

$lib.yaml.save(valu, sort_keys=$lib.true)

Encode data as a YAML string.

Args:
valu (object): The object to encode.

sort_keys (boolean): Sort object keys.

Returns:
A YAML string. The return type is *str*.

13.2 Storm Types

Storm Objects are used as view objects for manipulating data in the Storm Runtime and in the Cortex itself.

13.2.1 aha:pool

Implements the Storm API for an AHA pool.

add(svcname)

Add a service to the AHA pool

Examples:
Add a service to a pool with its relative name:

```
$pool = $lib.aha.pool.get(pool00.cortex...)
$pool.add(00.cortex...)
```

Args:
svcname (str): The name of the AHA service to add. It is easiest to use the relative name of a service, ending with "…".

Returns:
The return type is `null`.

add(svcname)

Remove a service from the AHA pool.

Examples:
Remove a service from a pool with its relative name:

```
$pool = $lib.aha.pool.get(pool00.cortex...)
$pool.del(00.cortex...)
```

Args:
svcname (str): The name of the AHA service to remove. It is easiest to use the relative name of a service, ending with "...".

Returns:
The return type is `null`.

13.2.2 auth:gate

Implements the Storm API for an AuthGate.

iden

The iden of the AuthGate.

Returns:
The type is *str*.

roles

The role idens which are a member of the Authgate.

Returns:
The type is *list*.

type

The type of the AuthGate.

Returns:
The type is *str*.

users

The user idens which are a member of the Authgate.

Returns:
The type is *list*.

13.2.3 auth:role

Implements the Storm API for a Role.

addRule(rule, gateiden=$lib.null, indx=$lib.null)

Add a rule to the Role

Args:
rule (list): The rule tuple to added to the Role.

gateiden (str): The gate iden used for the rule.

indx (int): The position of the rule as a 0 based index.

Returns:
The return type is `null`.

delRule(rule, gateiden=$lib.null)

Remove a rule from the Role.

Args:
rule (list): The rule tuple to removed from the Role.

gateiden (str): The gate iden used for the rule.

Returns:
The return type is `null`.

gates()

Return a list of auth gates that the role has rules for.

Returns:
A list of `auth:gates` that the role has rules for. The return type is *list*.

get(name)

Get a arbitrary property from the Role definition.

Args:
name (str): The name of the property to return.

Returns:
The requested value. The return type is `prim`.

getRules(gateiden=$lib.null)

Get the rules for the role and optional auth gate.

Args:
  gateiden (str): The gate iden used for the rules.

Returns:
  A list of rules. The return type is *list*.

iden

The Role iden.

Returns:
  The type is *str*.

name

A role's name. This can also be used to set the role name.

Example:
  Change a role's name:

```
$role=$lib.auth.roles.byname(analyst) $role.name=superheroes
```

Returns:
  The return type is *str*. When this is used to set the value, it does not have a return type.

pack()

Get the packed version of the Role.

Returns:
  The packed Role definition. The return type is *dict*.

popRule(indx, gateiden=$lib.null)

Remove a rule by index from the Role.

Args:
  indx (int): The index of the rule to remove.

  gateiden (str): The gate iden used for the rule.

Returns:
  The rule which was removed. The return type is *list*.

setRules(rules, gateiden=$lib.null)

Replace the rules on the Role with new rules.

Args:
: rules (list): A list of rules to set on the Role.

  gateiden (str): The gate iden used for the rules.

Returns:
: The return type is `null`.

13.2.4 auth:user

Implements the Storm API for a User.

addRule(rule, gateiden=$lib.null, indx=$lib.null)

Add a rule to the User.

Args:
: rule (list): The rule tuple to add to the User.

  gateiden (str): The gate iden used for the rule.

  indx (int): The position of the rule as a 0 based index.

Returns:
: The return type is `null`.

allowed(permname, gateiden=$lib.null, default=$lib.false)

Check if the user has a given permission.

Args:
: permname (str): The permission string to check.

  gateiden (str): The authgate iden.

  default (boolean): The default value.

Returns:
: True if the rule is allowed, False otherwise. The return type is *boolean*.

delApiKey(iden)

Delete an existing API key.

Args:
: iden (str): The iden of the API key.

Returns:
: True when the key was deleted. The return type is *boolean*.

delRule(rule, gateiden=$lib.null)

Remove a rule from the User.

Args:
rule (list): The rule tuple to removed from the User.

gateiden (str): The gate iden used for the rule.

Returns:
The return type is `null`.

email

A user's email. This can also be used to set the user's email.

Example:
Change a user's email address:

```
$user=$lib.auth.users.byname(bob) $user.email="robert@bobcorp.net"
```

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

gates()

Return a list of auth gates that the user has rules for.

Returns:
A list of `auth:gates` that the user has rules for. The return type is *list*.

genApiKey(name, duration=$lib.null)

Generate a new API key for the user.

Notes:
The secret API key returned by this function cannot be accessed again.

Args:
name (str): The name of the API key.

duration (integer): Duration of time for the API key to be valid, in milliseconds.

Returns:
A list, containing the secret API key and a dictionary containing metadata about the key. The return type is *list*.

get(name)

Get a arbitrary property from the User definition.

Args:
name (str): The name of the property to return.

Returns:
The requested value. The return type is `prim`.

getAllowedReason(permname, gateiden=$lib.null, default=$lib.false)

Return an allowed status and reason for the given perm.

Args:
permname (str): The permission string to check.

gateiden (str): The authgate iden.

default (boolean): The default value.

Returns:
An (allowed, reason) tuple. The return type is *list*.

getApiKey(iden)

Get information about a user's existing API key.

Args:
iden (str): The iden of the API key.

Returns:
A dictionary containing metadata about the key. The return type is *dict*.

getRules(gateiden=$lib.null)

Get the rules for the user and optional auth gate.

Args:
gateiden (str): The gate iden used for the rules.

Returns:
A list of rules. The return type is *list*.

grant(iden, indx=$lib.null)

Grant a Role to the User.

Args:
iden (str): The iden of the Role.

indx (int): The position of the Role as a 0 based index.

Returns:
The return type is `null`.

iden

The User iden.

Returns:
The type is *str*.

listApiKeys()

Get information about all the API keys the user has.

Returns:
A list of dictionaries containing metadata about each key. The return type is *list*.

modApiKey(iden, name, valu)

Modify metadata about an existing API key.

Args:
iden (str): The iden of the API key.

name (str): The name of the valu to update.

valu (any): The new value of the API key.

Returns:
An updated dictionary with metadata about the key. The return type is *dict*.

name

A user's name. This can also be used to set a user's name.

Example:
Change a user's name:

```
$user=$lib.auth.users.byname(bob) $user.name=robert
```

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

notify(mesgtype, mesgdata)

Send an arbitrary user notification.

Args:
mesgtype (str): The notification type.

mesgdata (dict): The notification data.

Returns:
The return type is `null`.

pack()

Get the packed version of the User.

Returns:
The packed User definition. The return type is *dict*.

popRule(indx, gateiden=$lib.null)

Remove a rule by index from the User.

Args:
indx (int): The index of the rule to remove.

gateiden (str): The gate iden used for the rule.

Returns:
The rule which was removed. The return type is *list*.

profile

A user profile dictionary. This can be used as an application level key-value store.

Example:
Set a value:

```
$user=$lib.auth.users.byname(bob) $user.profile.somekey="somevalue"
```

Get a value:

```
$user=$lib.auth.users.byname(bob) $value = $user.profile.somekey
```

Returns:
The return type is *auth:user:profile*.

revoke(iden)

Remove a Role from the User

Args:
iden (str): The iden of the Role.

Returns:
The return type is `null`.

roles()

Get the Roles for the User.

Returns:
A list of `auth:roles` which the user is a member of. The return type is *list*.

setAdmin(admin, gateiden=$lib.null)

Set the Admin flag for the user.

Args:
admin (boolean): True to make the User an admin, false to remove their admin status.

gateiden (str): The gate iden used for the operation.

Returns:
The return type is `null`.

setEmail(email)

Set the email address of the User.

Args:
email (str): The email address to set for the User.

Returns:
The return type is `null`.

setLocked(locked)

Set the locked status for a user.

Args:
locked (boolean): True to lock the user, false to unlock them.

Returns:
The return type is `null`.

setPasswd(passwd)

Set the Users password.

Args:
passwd (str): The new password for the user. This is best passed into the runtime as a variable.

Returns:
The return type is `null`.

setRoles(idens)

Replace all the Roles of the User with a new list of roles.

Notes:
The roleiden for the "all" role must be present in the new list of roles. This replaces all existing roles that the user has with the new roles.

Args:
idens (list): The idens to of the Role.

Returns:
The return type is `null`.

setRules(rules, gateiden=$lib.null)

Replace the rules on the User with new rules.

Args:
rules (list): A list of rule tuples.

gateiden (str): The gate iden used for the rules.

Returns:
The return type is `null`.

tell(text)

Send a tell notification to a user.

Args:
text (str): The text of the message to send.

Returns:
The return type is `null`.

vars

Get a dictionary representing the user's persistent variables.

Returns:
The return type is *auth:user:vars*.

13.2.5 auth:user:json

Implements per-user JSON storage.

del(path, prop=$lib.null)

Delete a stored JSON object or object property for the user.

Args:
path (str|list): A path string or list of path parts.

prop (str|list): A property name or list of name parts.

Returns:
True if the del operation was successful. The return type is *boolean*.

get(path, prop=$lib.null)

Return a stored JSON object or object property for the user.

Args:
path (str|list): A path string or list of path parts.

prop (str|list): A property name or list of name parts.

Returns:
The previously stored value or $lib.null The return type is `prim`.

iter(path=$lib.null)

Yield (<path>, <valu>) tuples for the users JSON objects.

Args:
path (str|list): A path string or list of path parts.

Yields:
(<path>, <item>) tuples. The return type is *list*.

set(path, valu, prop=$lib.null)

Set a JSON object or object property for the user.

Args:
path (str|list): A path string or list of path elements.

valu (prim): The value to set as the JSON object or object property.

prop (str|list): A property name or list of name parts.

Returns:
True if the set operation was successful. The return type is *boolean*.

13.2.6 auth:user:profile

The Storm deref/setitem/iter convention on top of User profile information.

13.2.7 auth:user:vars

The Storm deref/setitem/iter convention on top of User vars information.

13.2.8 boolean

Implements the Storm API for a boolean instance.

13.2.9 bytes

Implements the Storm API for a Bytes object.

bunzip()

Decompress the bytes using bzip2.

Example:
Decompress bytes with bzip2:

```
$foo = $mybytez.bunzip()
```

Returns:
Decompressed bytes. The return type is *bytes*.

bzip()

Compress the bytes using bzip2 and return them.

Example:
Compress bytes with bzip:

```
$foo = $mybytez.bzip()
```

Returns:
The bzip2 compressed bytes. The return type is *bytes*.

decode(encoding=utf8, errors=surrogatepass)

Decode bytes to a string.

Args:
encoding (str): The encoding to use.

errors (str): The error handling scheme to use.

Returns:
The decoded string. The return type is *str*.

gunzip()

Decompress the bytes using gzip and return them.

Example:
Decompress bytes with bzip2:

```
$foo = $mybytez.gunzip()
```

Returns:
Decompressed bytes. The return type is *bytes*.

gzip()

Compress the bytes using gzip and return them.

Example:
Compress bytes with gzip:

```
$foo = $mybytez.gzip()
```

Returns:
The gzip compressed bytes. The return type is *bytes*.

json(encoding=$lib.null, errors=surrogatepass)

Load JSON data from bytes.

Notes:
The bytes must be UTF8, UTF16 or UTF32 encoded.

Example:
Load bytes to a object:

```
$foo = $mybytez.json()
```

Args:
encoding (str): Specify an encoding to use.

errors (str): Specify an error handling scheme to use.

Returns:
The deserialized object. The return type is `prim`.

slice(start, end=$lib.null)

Slice a subset of bytes from an existing bytes.

Examples:
Slice from index to 1 to 5:

```
$subbyts = $byts.slice(1,5)
```

Slice from index 3 to the end of the bytes:

```
$subbyts = $byts.slice(3)
```

Args:
start (int): The starting byte index.

end (int): The ending byte index. If not specified, slice to the end.

Returns:
The slice of bytes. The return type is *bytes*.

unpack(fmt, offset=(0))

Unpack structures from bytes using python struct.unpack syntax.

Examples:
Unpack 3 unsigned 16 bit integers in little endian format:

```
($x, $y, $z) = $byts.unpack("<HHH")
```

Args:
fmt (str): A python struck.pack format string.

offset (int): An offset to begin unpacking from.

Returns:
The unpacked primitive values. The return type is *list*.

13.2.10 cache:fixed

A StormLib API instance of a Storm Fixed Cache.

clear()

Clear all items from the cache.

Returns:
The return type is `null`.

get(key)

Get an item from the cache by key.

Args:
key (any): The key to lookup.

Returns:
The value from the cache, or the callback query if it does not exist The return type is `any`.

pop(key)

Pop an item from the cache.

Args:
key (any): The key to pop.

Returns:
The value from the cache, or $lib.null if it does not exist The return type is `any`.

put(key, value)

Put an item into the cache.

Args:
key (any): The key put in the cache.

value (any): The value to assign to the key.

Returns:
The return type is `null`.

query

Get the callback Storm query as string.

Returns:
The callback Storm query text. The return type is *str*.

13.2.11 cmdopts

A dictionary like object that holds a reference to a command options namespace. (This allows late-evaluation of command arguments rather than forcing capture)

13.2.12 cronjob

Implements the Storm api for a cronjob instance.

iden

The iden of the Cron Job.

Returns:
The type is *str*.

pack()

Get the Cronjob definition.

Returns:
The definition. The return type is *dict*.

pprint()

Get a dictionary containing user friendly strings for printing the CronJob.

Returns:
A dictionary containing structured data about a cronjob for display purposes. The return type is *dict*.

set(name, valu)

Set an editable field in the cron job definition.

Example:
Change the name of a cron job:

```
$lib.cron.get($iden).set(name, "foo bar cron job")
```

Args:
name (str): The name of the field being set

valu (any): The value to set on the definition.

Returns:
The `cronjob` The return type is *cronjob*.

13.2.13 dict

Implements the Storm API for a Dictionary object.

13.2.14 hive:dict

A Storm Primitive representing a HiveDict.

get(name, default=$lib.null)

Get the named value from the HiveDict.

Args:
name (str): The name of the value.

default (prim): The default value to return if the name is not set.

Returns:
The requested value. The return type is `prim`.

list()

List the keys and values in the HiveDict.

Returns:
A list of tuples containing key, value pairs. The return type is *list*.

pop(name, default=$lib.null)

Remove a value out of the HiveDict.

Args:
name (str): The name of the value.

default (prim): The default value to return if the name is not set.

Returns:
The requested value. The return type is `prim`.

set(name, valu)

Set a value in the HiveDict.

Args:
name (str): The name of the value to set

valu (prim): The value to store in the HiveDict

Returns:
Old value of the dictionary if the value was previously set, or none. The return type may be one of the following: `null`, `prim`.

13.2.15 http:api

Extended HTTP API object.

This object represents an extended HTTP API that has been configured on the Cortex.

authenticated

Boolean value indicating if the Extended HTTP API requires an authenticated user or session.

Returns:
The return type is *boolean*. When this is used to set the value, it does not have a return type.

created

The time the Extended HTTP API was created.

Returns:
The type is `int`.

creator

The user that created the Extended HTTP API.

Returns:
The return type is *auth:user*.

desc

The description of the API instance.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

iden

The iden of the Extended HTTP API.

Returns:
The type is *str*.

methods

The dictionary containing the Storm code used to implement the HTTP methods.

Returns:
The return type is *http:api:methods*.

name

The name of the API instance.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

owner

The user that runs the endpoint query logic when runas="owner".

Returns:
The return type is *auth:user*. When this is used to set the value, it does not have a return type.

pack()

Get a packed copy of the HTTP API object.

Returns:
The return type is *dict*.

path

The path of the API instance.

This path may contain regular expression capture groups, which are used to populate request arguments.

Note:
The Cortex does not inspect paths in order to identify duplicates or overlapping paths. It is the responsibility of the Cortex administrator to configure their Extended HTTP API paths so that they are correct for their use cases.

Example:
Update an API path to contain a single wildcard argument:

```
$api.path = 'foo/bar/(.*)/baz'
```

Update an API path to contain a two wildcard arguments with restricted character sets:

```
$api.path = 'hehe/([a-z]*)/([0-9]{1-4})'
```

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

perms

The permissions an authenticated user must have in order to access the HTTP API.

Returns:
The return type is *http:api:perms*. When this is used to set the value, it does not have a return type.

pool

Boolean value indicating if the handler responses may be executed as part of a Storm pool.

Returns:
The return type is *boolean*. When this is used to set the value, it does not have a return type.

readonly

Boolean value indicating if the Storm methods are executed in a readonly Storm runtime.

Returns:
The return type is *boolean*. When this is used to set the value, it does not have a return type.

runas

String indicating whether the requests run as the owner or the authenticated user.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

updated

The time the Extended HTTP API was last modified.

Returns:
The return type is `int`.

vars

The Storm runtime variables specific for the API instance.

Returns:
The return type is *http:api:vars*. When this is used to set the value, it does not have a return type.

view

The View of the API instance. This is the view that Storm methods are executed in.

Returns:
The return type is *view*. When this is used to set the value, it does not have a return type.

13.2.16 http:api:methods

Accessor dictionary for getting and setting Extended HTTP API methods.

Notes:
The Storm code used to run these methods will have a $request object injected into them. This allows the method to send data back to the caller when it is run.

Examples:
Setting a simple GET method:

```
$api.methods.get = ${
    $data = ({"someKey": "someValue})
    $headers = ({"someHeader": "someOtherValue"})
    $request.reply(200, headers=$headers, body=$data)
}
```

Removing a PUT method:

```
$api.methods.put = $lib.undef
```

Crafting a custom text response:

```
$api.methods.get = ${
    // Create the body
    $data = 'some value'
    // Encode the response as bytes
    $data = $data.encode()
    // Set the headers
    $headers = ({"Content-Type": "text/plain", "Content-Length": $lib.len($data})
    $request.reply(200, headers=$headers, body=$data)
}
```

Streaming multiple chunks of data as JSON lines. This sends the code, headers and body separately:

```
$api.methods.get = ${
    $request.sendcode(200)
    $request.sendheaders(({"Content-Type": "text/plain; charset=utf8"}))
    $values = ((1), (2), (3))
    for $i in $values {
        $body=`{$lib.json.save(({"value": $i}))}

        $request.sendbody($body.encode())
    }
}
```

delete

The DELETE request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

get

The GET request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

head

The HEAD request Storm code

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

options

The OPTIONS request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

patch

The PATCH request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

post

The POST request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

put

The PUT request Storm code.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

13.2.17 http:api:perms

Accessor list for getting and setting http:api permissions.

append(valu)

Append a permission to the list.

Args:
valu (any): The permission to append to the list.

Returns:
The return type is `null`.

extend(valu)

Extend a list using another iterable.

Args:
valu (list): A list or other iterable.

Returns:
The return type is `null`.

has(valu)

Check it a permission is in the list.

Args:
valu (any): The permission to check.

Returns:
True if the permission is in the list, false otherwise. The return type is *boolean*.

index(valu)

Return a single permission from the list by index.

Args:
valu (int): The list index value.

Returns:
The permission present in the list at the index position. The return type is `any`.

length()

Get the length of the list. This is deprecated; please use `.size()` instead.

Returns:
The size of the list. The return type is `int`.

pop()

Pop and return the last permission in the list.

Returns:
The last permission from the list. The return type is `any`.

reverse()

Reverse the order of the list in place

Returns:
The return type is `null`.

size()

Return the length of the list.

Returns:
The size of the list. The return type is `int`.

slice(start, end=$lib.null)

Get a slice of the list.

Args:
start (int): The starting index.

end (int): The ending index. If not specified, slice to the end of the list.

Returns:
The slice of the list. The return type is *list*.

13.2.18 http:api:request

Extended HTTP API Request object.

api

The http:api object for the request.

Returns:
The return type is *http:api*.

args

A list of path arguments made as part of the HTTP API request. These are the results of any capture groups defined in the Extended HTTP API path regular expression.

Returns:
The type is *list*.

body

The raw request body.

Returns:
The type is *bytes*.

client

The remote IP of the requester.

Returns:
The type is *str*.

headers

The request headers.

Returns:
The return type is *http:api:request:headers*.

json

The request body as json.

Returns:
The return type is *dict*.

method

The request method

Returns:
The type is *str*.

params

Request parameters.

Returns:
The type is *dict*.

path

The path which was matched against the Extended HTTPAPI endpoint.

Returns:
The type is *str*.

reply(code, headers=$lib.null, body=$lib.undef)

Convenience method to send the response code, headers and body together.

Notes:
This can only be called once.

If the response body is not bytes, this method will serialize the body as JSON and set the `Content-Type` and `Content-Length` response headers.

Args:
code (int): The response code.

headers (dict): The response headers.

body (any): The response body.

Returns:
The return type is `null`.

sendbody(body)

Send the HTTP response body.

Args:
body (bytes): The response body.

Returns:
The return type is `null`.

sendcode(code)

Send the HTTP response code.

Args:
code (int): The response code.

Returns:
The return type is `null`.

sendheaders(headers)

Send the HTTP response headers.

Args:
headers (dict): The response headers.

Returns:
The return type is `null`.

uri

The full request URI.

Returns:
The type is *str*.

user

The user iden who made the HTTP API request.

Returns:
The type is *str*.

13.2.19 http:api:request:headers

Immutable lowercase key access dictionary for HTTP request headers.

Example:
Request headers can be accessed in a case insensitive manner:

```
$valu = $request.headers.Cookie
// or the lower case value
$valu = $request.headers.cookie
```

13.2.20 http:api:vars

Accessor dictionary for getting and setting Extended HTTP API variables.

This can be used to set, unset or iterate over the runtime variables that are set for an Extended HTTP API endpoint. These variables are set in the Storm runtime for all of the HTTP methods configured to be executed by the endpoint.

Example:
Set a few variables on a given API:

```
$api.vars.foo = 'the foo string'
$api.vars.bar = (1234)
```

Remove a variable:

```
$api.vars.foo = $lib.undef
```

Iterate over the variables set for the endpoint:

```
for ($key, $valu) in $api.vars {
    $lib.print(`{$key) -> {$valu}`)
}
```

Overwrite all of the variables for a given API with a new dictionary:

```
$api.vars = ({"foo": "a new string", "bar": (137)})
```

13.2.21 inet:http:oauth:v1:client

A client for doing OAuth V1 Authentication from Storm.

sign(baseurl, method=GET, headers=$lib.null, params=$lib.null, body=$lib.null)

Sign an OAuth request to a particular URL.

Args:

baseurl (str): The base url to sign and query.

method (dict): The HTTP Method to use as part of signing.

headers (dict): Optional headers used for signing. Can override the "Content-Type" header if the signature type is set to SIG_BODY

params (dict): Optional query parameters to pass to url construction and/or signing.

body (bytes): Optional HTTP body to pass to request signing.

Returns:

A 3-element tuple of ($url, $headers, $body). The OAuth signature elements will be embedded in the element specified when constructing the client. The return type is *list*.

13.2.22 inet:http:resp

Implements the Storm API for a HTTP response.

body

The raw HTTP response body as bytes.

Returns:

The type is *bytes*.

code

The HTTP status code. It is -1 if an exception occurred.

Returns:

The type is `int`.

err

Tuple of the error type and information if an exception occurred.

Returns:
The type is *list*.

headers

The HTTP Response headers.

Returns:
The type is *dict*.

json(encoding=$lib.null, errors=surrogatepass)

Get the JSON deserialized response.

Args:
encoding (str): Specify an encoding to use.

errors (str): Specify an error handling scheme to use.

Returns:
The return type is `prim`.

msgpack()

Yield the msgpack deserialized objects.

Yields:
Unpacked values. The return type is `prim`.

reason

The reason phrase for the HTTP status code.

Returns:
The type is *str*.

13.2.23 inet:http:socket

Implements the Storm API for a Websocket.

rx(timeout=$lib.null)

Receive a message from the web socket.

Args:
timeout (int): The timeout to wait for

Returns:
An ($ok, $valu) tuple. The return type is *list*.

tx(mesg)

Transmit a message over the web socket.

Args:
mesg (dict): A JSON compatible message.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

13.2.24 inet:imap:server

An IMAP server for retrieving email messages.

delete(uid_set)

Mark an RFC2060 UID message as deleted and expunge the mailbox.

The command uses the +FLAGS.SILENT command and applies the Deleted flag. The actual behavior of these commands are mailbox configuration dependent.

Examples:
Mark a single message as deleted and expunge:

```
($ok, $valu) = $server.delete("8182")
```

Mark ranges of messages as deleted and expunge:

```
($ok, $valu) = $server.delete("1:3,6:9")
```

Args:
uid_set (str): The UID message set to apply the flag to.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

fetch(uid)

Fetch a message by UID in RFC822 format.

The message is saved to the Axon, and a `file:bytes` node is returned.

Examples:
Fetch a message, save to the Axon, and yield `file:bytes` node:

```
yield $server.fetch("8182")
```

Args:
uid (str): The single message UID.

Returns:
The file:bytes node representing the message. The return type is *node*.

list(reference_name="", pattern=*)

List mailbox names.

By default this method uses a reference_name and pattern to return all mailboxes from the root.

Args:
reference_name (str): The mailbox reference name.

pattern (str): The pattern to filter by.

Returns:
An ($ok, $valu) tuple where $valu is a list of names if $ok=True. The return type is *list*.

login(user, passwd)

Login to the IMAP server.

Args:
user (str): The username to login with.

passwd (str): The password to login with.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

markSeen(uid_set)

Mark messages as seen by an RFC2060 UID message set.

The command uses the +FLAGS.SILENT command and applies the Seen flag.

Examples:
Mark a single messsage as seen:

```
($ok, $valu) = $server.markSeen("8182")
```

Mark ranges of messages as seen:

```
($ok, $valu) = $server.markSeen("1:3,6:9")
```

Args:
uid_set (str): The UID message set to apply the flag to.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

search(*args, charset=utf-8)

Search for messages using RFC2060 syntax.

Examples:
Retrieve all messages:

```
($ok, $uids) = $server.search("ALL")
```

Search by FROM and SINCE:

```
($ok, $uids) = $server.search("FROM", "visi@vertex.link", "SINCE", "01-Oct-2021")
```

Search by a subject substring:

```
($ok, $uids) = $search.search("HEADER", "Subject", "An email subject")
```

Args:
*args (str): A set of search criteria to use.

charset: The CHARSET used for the search. May be set to $lib.null to disable CHARSET. The input type may be one of the following: `str`, `null`.

Returns:
An ($ok, $valu) tuple, where $valu is a list of UIDs if $ok=True. The return type is *list*.

select(mailbox=INBOX)

Select a mailbox to use in subsequent commands.

Args:
mailbox (str): The mailbox name to select.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

13.2.25 inet:smtp:message

An SMTP message to compose and send.

headers

A dictionary of email header values.

Returns:
The type is *dict*.

html

The HTML body of the email message. This can also be used to set an HTML body in the message.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

recipients

An array of RCPT TO email addresses.

Returns:
The type is *list*.

send(host, port=(25), user=$lib.null, passwd=$lib.null, usetls=$lib.false, starttls=$lib.false, timeout=(60))

Transmit a message over the web socket.

Args:
host (str): The hostname or IP address of the SMTP server.

port (int): The port that the SMTP server is listening on.

user (str): The user name to use authenticating to the SMTP server.

passwd (str): The password to use authenticating to the SMTP server.

usetls (bool): Initiate a TLS connection to the SMTP server.

starttls (bool): Use the STARTTLS directive with the SMTP server.

timeout (int): The timeout (in seconds) to wait for message delivery.

Returns:
An ($ok, $valu) tuple. The return type is *list*.

sender

The inet:email to use in the MAIL FROM request. This can also be used to set the sender for the message.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

text

The text body of the email message. This can also be used to set the body of the message.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

13.2.26 json:schema

A JsonSchema validation object for use in validating data structures in Storm.

schema()

The schema belonging to this object.

Returns:
A copy of the schema used for this object. The return type is *dict*.

validate(item)

Validate a structure against the Json Schema

Args:
item (prim): A JSON structure to validate (dict, list, etc...)

Returns:
An ($ok, $valu) tuple. If $ok is True, then $valu should be used as the validated data structure. If $ok is False, $valu is a dictionary with a "mesg" key. The return type is *list*.

13.2.27 layer

Implements the Storm api for a layer instance.

addPull(url, offs=(0), queue_size=(10000), chunk_size=(1000))

Configure the layer to pull edits from a remote layer/feed.

Args:
url (str): The telepath URL to a layer/feed.

offs (int): The offset to begin from.

queue_size (int): The queue size of the puller.

chunk_size (int): The chunk size of the puller when consuming edits.

Returns:
Dictionary containing the pull definition. The return type is *dict*.

addPush(url, offs=(0), queue_size=(10000), chunk_size=(1000))

Configure the layer to push edits to a remote layer/feed.

Args:
url (str): A telepath URL of the target layer/feed.

offs (int): The local layer offset to begin pushing from

queue_size (int): The queue size of the pusher.

chunk_size (int): The chunk size of the pusher when pushing edits.

Returns:
Dictionary containing the push definition. The return type is *dict*.

delPull(iden)

Remove a pull config from the layer.

Args:
iden (str): The iden of the push config to remove.

Returns:
The return type is `null`.

delPush(iden)

Remove a push config from the layer.

Args:
iden (str): The iden of the push config to remove.

Returns:
The return type is `null`.

edits(offs=(0), wait=$lib.true, size=$lib.null)

Yield (offs, nodeedits) tuples from the given offset.

Args:
offs (int): Offset to start getting nodeedits from the layer at.

wait (boolean): If true, wait for new edits, otherwise exit the generator when there are no more edits.

size (int): The maximum number of nodeedits to yield.

Yields:
Yields offset, nodeedit tuples from a given offset. The return type is *list*.

get(name, defv=$lib.null)

Get a arbitrary value in the Layer definition.

Args:
name (str): Name of the value to get.

defv (prim): The default value returned if the name is not set in the Layer.

Returns:
The value requested or the default value. The return type is `prim`.

getEdges()

Yield (n1iden, verb, n2iden) tuples for any light edges in the layer.

Example:
Iterate the light edges in `$layer`:

```
for ($n1iden, $verb, $n2iden) in $layer.getEdges() {
    $lib.print(`{$n1iden} -({$verb})> {$n2iden}`)
}
```

Yields:
Yields (<n1iden>, <verb>, <n2iden>) tuples The return type is *list*.

getEdgesByN1(nodeid)

Yield (verb, n2iden) tuples for any light edges in the layer for the source node id.

Example:
Iterate the N1 edges for `$node`:

```
for ($verb, $n2iden) in $layer.getEdgesByN1($node.iden()) {
    $lib.print(`-({$verb})> {$n2iden}`)
}
```

Args:
nodeid (str): The hex string of the node id.

Yields:
Yields (<verb>, <n2iden>) tuples The return type is *list*.

getEdgesByN2(nodeid)

Yield (verb, n1iden) tuples for any light edges in the layer for the target node id.

Example:
Iterate the N2 edges for `$node`:

```
for ($verb, $n1iden) in $layer.getEdgesByN2($node.iden()) {
    $lib.print(`-({$verb})> {$n1iden}`)
}
```

Args:
nodeid (str): The hex string of the node id.

Yields:
Yields (<verb>, <n1iden>) tuples The return type is *list*.

getFormCounts()

Get the formcounts for the Layer.

Example:
Get the formcounts for the current Layer:

```
$counts = $lib.layer.get().getFormCounts()
```

Returns:
Dictionary containing form names and the count of the nodes in the Layer. The return type is *dict*.

getMirrorStatus()

Return a dictionary of the mirror synchronization status for the layer.

Returns:
An info dictionary describing mirror sync status. The return type is *dict*.

getPropArrayCount(propname, valu=$lib.undef)

Get the number of individual value rows in the layer for the given array property name.

Args:
propname (str): The property name to look up.

valu (any): A specific value in the array property to look up.

Returns:
The count of rows. The return type is `int`.

getPropCount(propname, maxsize=$lib.null, valu=$lib.undef)

Get the number of property rows in the layer for the given full form or property name.

Args:
propname (str): The property or form name to look up.

maxsize (int): The maximum number of rows to look up.

valu (any): A specific value of the property to look up.

Returns:
The count of rows. The return type is `int`.

getStorNode(nodeid)

Retrieve the raw storage node for the specified node id.

Args:
nodeid (str): The hex string of the node id.

Returns:
The storage node dictionary. The return type is *dict*.

getStorNodes()

Get buid, sode tuples representing the data stored in the layer.

Notes:
The storage nodes represent **only** the data stored in the layer and may not represent whole nodes.

Yields:
Tuple of buid, sode values. The return type is *list*.

getTagCount(tagname, formname=$lib.null)

Return the number of tag rows in the layer for the given tag and optional form.

Examples:
Get the number of `inet:ipv4` nodes with the `$foo.bar` tag:

```
$count = $lib.layer.get().getTagCount(foo.bar, formname=inet:ipv4)
```

Args:
tagname (str): The name of the tag to look up.

formname (str): The form to constrain the look up by.

Returns:
The count of tag rows. The return type is `int`.

getTagPropCount(tag, propname, form=$lib.null, valu=$lib.undef)

Get the number of rows in the layer for the given tag property.

Args:
tag (str): The tag to look up.

propname (str): The property name to look up.

form (str): The optional form to look up.

valu (any): A specific value of the property to look up.

Returns:
The count of rows. The return type is `int`.

iden

The iden of the Layer.

Returns:
The type is *str*.

liftByProp(propname, propvalu=$lib.null, propcmpr==)

Lift and yield nodes with the property and optional value set within the layer.

Example:
Yield all nodes with the property `ou:org:name` set in the top layer:

```
yield $lib.layer.get().liftByProp(ou:org:name)
```

Yield all nodes with the property `ou:org:name=woot` in the top layer:

```
yield $lib.layer.get().liftByProp(ou:org:name, woot)
```

Yield all nodes with the property `ou:org:name^=woot` in the top layer:

```
yield $lib.layer.get().liftByProp(ou:org:name, woot, "^=")
```

Args:
propname (str): The full property name to lift by.

propvalu (obj): The value for the property.

propcmpr (str): The comparison operation to use on the value.

Yields:
Yields nodes. The return type is *node*.

liftByTag(tagname, formname=$lib.null)

Lift and yield nodes with the tag set within the layer.

Example:
Yield all nodes with the tag #foo set in the layer:

```
yield $lib.layer.get().liftByTag(foo)
```

Yield all inet:fqdn with the tag #foo set in the layer:

```
yield $lib.layer.get().liftByTag(foo, inet:fqdn)
```

Args:
tagname (str): The tag name to lift by.

formname (str): The optional form to lift.

Yields:
Yields nodes. The return type is *node*.

pack()

Get the Layer definition.

Returns:
Dictionary containing the Layer definition. The return type is *dict*.

repr()

Get a string representation of the Layer.

Returns:
A string that can be printed, representing a Layer. The return type is *str*.

set(name, valu)

Set an arbitrary value in the Layer definition.

Args:
name (str): The name to set.

valu (any): The value to set.

Returns:
The return type is `null`.

verify(config=$lib.null)

Verify consistency between the node storage and indexes in the given layer.

Example:
Get all messages about consistency issues in the default layer:

```
for $mesg in $lib.layer.get().verify() {
    $lib.print($mesg)
}
```

Notes:
The config format argument and message format yielded by this API is considered BETA and may be subject to change! The formats will be documented when the convention stabilizes.

Args:
config (dict): The scan config to use (default all enabled).

Yields:
Yields messages describing any index inconsistencies. The return type is *list*.

13.2.28 list

Implements the Storm API for a List instance.

append(valu)

Append a value to the list.

Args:
valu (any): The item to append to the list.

Returns:
The return type is `null`.

extend(valu)

Extend a list using another iterable.

Examples:
Populate a list by extending it with to other lists:

```
$list = $lib.list()

$foo = (f, o, o)
$bar = (b, a, r)

$list.extend($foo)
$list.extend($bar)

// $list is now (f, o, o, b, a, r)
```

Args:
valu (list): A list or other iterable.

Returns:
The return type is `null`.

has(valu)

Check if a value is in the list.

Args:
valu (any): The value to check.

Returns:
True if the item is in the list, false otherwise. The return type is *boolean*.

index(valu)

Return a single field from the list by index.

Args:
valu (int): The list index value.

Returns:
The item present in the list at the index position. The return type is `any`.

length()

Get the length of the list. This is deprecated; please use `.size()` instead.

Returns:
The size of the list. The return type is `int`.

pop()

Pop and return the last entry in the list.

Returns:
The last item from the list. The return type is `any`.

reverse()

Reverse the order of the list in place

Returns:
The return type is `null`.

size()

Return the length of the list.

Returns:
The size of the list. The return type is `int`.

slice(start, end=$lib.null)

Get a slice of the list.

Examples:
Slice from index to 1 to 5:

```
$x=(f, o, o, b, a, r)
$y=$x.slice(1,5)  // (o, o, b, a)
```

Slice from index 3 to the end of the list:

```
$y=$x.slice(3)  // (b, a, r)
```

Args:
start (int): The starting index.

end (int): The ending index. If not specified, slice to the end of the list.

Returns:
The slice of the list. The return type is *list*.

sort(reverse=$lib.false)

Sort the list in place.

Args:
reverse (bool): Sort the list in reverse order.

Returns:
The return type is `null`.

unique()

Get a copy of the list containing unique items.

Returns:
The return type is *list*.

13.2.29 model:form

Implements the Storm API for a Form.

name

The name of the Form

Returns:
The type is *str*.

prop(name)

Get a Property on the Form

Args:
name (str): The property to retrieve.

Returns:
The `model:property` instance if the property if present on the form or null. The return type may be one of the following: *model:property*, `null`.

type

Get the Type for the form.

Returns:
The return type is *model:type*.

13.2.30 model:property

Implements the Storm API for a Property.

form

Get the Form for the Property.

Returns:
The return type may be one of the following: *model:form*, `null`.

full

The full name of the Property.

Returns:
The type is *str*.

name

The short name of the Property.

Returns:
The type is *str*.

type

Get the Type for the Property.

Returns:
The return type is *model:type*.

13.2.31 model:tagprop

Implements the Storm API for a Tag Property.

name

The name of the Tag Property.

Returns:
The type is *str*.

type

Get the Type for the Tag Property.

Returns:
The return type is *model:type*.

13.2.32 model:type

A Storm types wrapper around a lib.types.Type

name

The name of the Type.

Returns:
The type is *str*.

norm(valu)

Get the norm and info for the Type.

Args:
valu (any): The value to norm.

Returns:
A tuple of the normed value and its information dictionary. The return type is *list*.

repr(valu)

Get the repr of a value for the Type.

Args:
valu (any): The value to get the repr of.

Returns:
The string form of the value as represented by the type. The return type is *str*.

stortype

The storetype of the Type.

Returns:
The type is `int`.

13.2.33 node

Implements the Storm api for a node instance.

addEdge(verb, iden)

Add a light-weight edge.

Args:
verb (str): The edge verb to add.

iden (str): The node id of the destination node.

Returns:
The return type is `null`.

delEdge(verb, iden)

Remove a light-weight edge.

Args:
verb (str): The edge verb to remove.

iden (str): The node id of the destination node to remove.

Returns:
The return type is `null`.

difftags(tags, prefix=$lib.null, apply=$lib.false, norm=$lib.false)

Get and optionally apply the difference between the current set of tags and another set.

Args:
tags (list): The set to compare against.

prefix (str): An optional prefix to match tags under.

apply (boolean): If true, apply the diff.

norm (boolean): Optionally norm the list of tags. If a prefix is provided, it will not be normed.

Returns:
The tags which have been added/deleted in the new set. The return type is *dict*.

edges(verb=$lib.null, reverse=$lib.false)

Yields the (verb, iden) tuples for this nodes edges.

Args:
    verb (str): If provided, only return edges with this verb.

    reverse (boolean): If true, yield edges with this node as the dest rather than source.

Yields:
    A tuple of (verb, iden) values for this nodes edges. The return type is *list*.

form()

Get the form of the Node.

Returns:
    The form of the Node. The return type is *str*.

getByLayer()

Return a dict you can use to lookup which props/tags came from which layers.

Returns:
    property / tag lookup dictionary. The return type is *dict*.

getStorNodes()

Return a list of "storage nodes" which were fused from the layers to make this node.

Returns:
    List of storage node objects. The return type is *list*.

globtags(glob)

Get a list of the tag components from a Node which match a tag glob expression.

Args:
    glob (str): The glob expression to match.

Returns:
    The components of tags which match the wildcard component of a glob expression. The return type is *list*.

iden()

Get the iden of the Node.

Returns:
    The nodes iden. The return type is *str*.

isform(name)

Check if a Node is a given form.

Args:
name (str): The form to compare the Node against.

Returns:
True if the form matches, false otherwise. The return type is *boolean*.

ndef()

Get the form and primary property of the Node.

Returns:
A tuple of the form and primary property. The return type is *list*.

pack(dorepr=$lib.false)

Return the serializable/packed version of the Node.

Args:
dorepr (boolean): Include repr information for human readable versions of properties.

Returns:
A tuple containing the ndef and property bag of the node. The return type is *list*.

repr(name=$lib.null, defv=$lib.null)

Get the repr for the primary property or secondary property of a Node.

Args:
name (str): The name of the secondary property to get the repr for.

defv (str): The default value to return if the secondary property does not exist

Returns:
The string representation of the requested value. The return type is *str*.

tags(glob=$lib.null, leaf=$lib.false)

Get a list of the tags on the Node.

Notes:
When providing a glob argument, the following rules are used. A single asterisk(*) will replace exactly one dot-delimited component of a tag. A double asterisk(**) will replace one or more of any character.

Args:
glob (str): A tag glob expression. If this is provided, only tags which match the expression are returned.

leaf (bool): If true, only leaf tags are included in the returned tags.

Returns:
A list of tags on the node. If a glob match is provided, only matching tags are returned. The return type is *list*.

value()

Get the value of the primary property of the Node.

Returns:
The primary property. The return type is `prim`.

13.2.34 node:data

A Storm Primitive representing the NodeData stored for a Node.

cacheget(name, asof=now)

Retrieve data stored with cacheset() if it was stored more recently than the asof argument.

Args:
name (str): The name of the data to load.

asof (time): The max cache age.

Returns:
The cached value or null. The return type is `prim`.

cacheset(name, valu)

Set a node data value with an envelope that tracks time for cache use.

Args:
name (str): The name of the data to set.

valu (prim): The data to store.

Returns:
The return type is `null`.

get(name)

Get the Node data for a given name for the Node.

Args:
name (str): Name of the data to get.

Returns:
The stored node data. The return type is `prim`.

has(name)

Check if the Node data has the given key set on it

Args:
name (str): Name of the data to check for.

Returns:
True if the key is found, otherwise false. The return type is *boolean*.

list()

Get a list of the Node data names on the Node.

Returns:
List of the names of values stored on the node. The return type is *list*.

load(name)

Load the Node data onto the Node so that the Node data is packed and returned by the runtime.

Args:
name (str): The name of the data to load.

Returns:
The return type is `null`.

pop(name)

Pop (remove) a the Node data from the Node.

Args:
name (str): The name of the data to remove from the node.

Returns:
The data removed. The return type is `prim`.

set(name, valu)

Set the Node data for a given name on the Node.

Args:
name (str): The name of the data.

valu (prim): The data to store.

Returns:
The return type is `null`.

13.2.35 node:path

Implements the Storm API for the Path object.

idens()

The list of Node idens which this Path has been forked from during pivot operations.

Returns:
A list of node idens. The return type is *list*.

listvars()

List variables available in the path of a storm query.

Returns:
List of tuples containing the name and value of path variables. The return type is *list*.

meta

The PathMeta object for the Path.

Returns:
The type is *node:path:meta*.

vars

The PathVars object for the Path.

Returns:
The type is *node:path:vars*.

13.2.36 node:path:meta

Put the storm deref/setitem/iter convention on top of path meta information.

13.2.37 node:path:vars

Put the storm deref/setitem/iter convention on top of path variables.

13.2.38 node:props

A Storm Primitive representing the properties on a Node.

get(name)

Get a specific property value by name.

Args:
name (str): The name of the property to return.

Returns:
The requested value. The return type is `prim`.

list()

List the properties and their values from the `$node`.

Returns:
A list of (name, value) tuples. The return type is *list*.

set(prop, valu)

Set a specific property value by name.

Args:
prop (str): The name of the property to set.

valu (prim): The value to set the property to.

Returns:
The set value. The return type is `prim`.

13.2.39 number

Implements the Storm API for a Number instance.

Storm Numbers are high precision fixed point decimals corresponding to the the hugenum storage type.

scaleb(other)

Return the number multiplied by 10**other.

Example:
Multiply the value by 10**-18:

```
$baz.scaleb(-18)
```

Args:
other (int): The amount to adjust the exponent.

Returns:
The exponent adjusted number. The return type is *number*.

tofloat()

Return the number as a float.

Returns:
The number as a float. The return type is `float`.

toint(rounding=$lib.null)

Return the number as an integer.

By default, decimal places will be truncated. Optionally, rounding rules can be specified by providing the name of a Python decimal rounding mode to the 'rounding' argument.

Example:
Round the value stored in $baz up instead of truncating:

```
$baz.toint(rounding=ROUND_UP)
```

Args:
rounding (str): An optional rounding mode to use.

Returns:
The number as an integer. The return type is `int`.

tostr()

Return the number as a string.

Returns:
The number as a string. The return type is *str*.

13.2.40 pipe

A Storm Pipe provides fast ephemeral queues.

put(item)

Add a single item to the Pipe.

Args:
item (any): An object to add to the Pipe.

Returns:
The return type is `null`.

puts(items)

Add a list of items to the Pipe.

Args:
items (list): A list of items to add.

Returns:
The return type is `null`.

size()

Retrieve the number of items in the Pipe.

Returns:
The number of items in the Pipe. The return type is `int`.

slice(size=(1000))

Return a list of up to size items from the Pipe.

Args:
size (int): The max number of items to return.

Returns:
A list of at least 1 item from the Pipe. The return type is *list*.

slices(size=(1000))

Yield lists of up to size items from the Pipe.

Notes:
The loop will exit when the Pipe is closed and empty.

Examples:
Operation on slices from a pipe one at a time:

```
for $slice in $pipe.slices(1000) {
    for $item in $slice { $dostuff($item) }
}
```

Operate on slices from a pipe in bulk:

```
for $slice in $pipe.slices(1000) {
    $dostuff_batch($slice)
}
```

Args:
size (int): The max number of items to yield per slice.

Yields:
Yields objects from the Pipe. The return type is `any`.

13.2.41 proj:comment

Implements the Storm API for a ProjectTicketComment

del()

Delete the comment.

Returns:
True if the ProjectTicketComment was deleted The return type is *boolean*.

text

The comment text. This can be used to set the text as well.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

13.2.42 proj:comments

Implements the Storm API for ProjectTicketComments objects, which are collections of comments associated with a ticket.

add(text)

Add a comment to the ticket.

Args:
text (str): The text for the new ProjectTicketComment.

Returns:
The newly created *proj:comment* object The return type is *proj:comment*.

get(guid)

Get a ticket comment by guid.

Args:
guid (str): The guid of the ProjectTicketComment to get.

Returns:
The *proj:comment* object The return type is *proj:comment*.

13.2.43 proj:epic

Implements the Storm API for a ProjectEpic

name

The name of the Epic. This can be used to set the name as well.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

13.2.44 proj:epics

Implements the Storm API for ProjectEpics objects, which are collections of ProjectEpic objects associated with a particular Project

add(name)

Add an epic.

Args:
name (str): The name for the new ProjectEpic.

Returns:
The newly created *proj:epic* object The return type is *proj:epic*.

del(name)

Delete an epic by name.

Args:
name (str): The name of the ProjectEpic to delete.

Returns:
True if the ProjectEpic can be found and deleted, otherwise False The return type is *boolean*.

get(name)

Get an epic by name.

Args:
name (str): The name (or iden) of the ProjectEpic to get.

Returns:
The *proj:epic* object The return type is *proj:epic*.

13.2.45 proj:project

Implements the Storm API for Project objects, which are used for managing a scrum style project in the Cortex

epics

A *proj:epics* object that contains the epics associated with the given project.

Returns:
The return type is *proj:epics*.

name

The name of the project. This can also be used to set the name of the project.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

sprints

A *proj:sprints* object that contains the sprints associated with the given project.

Returns:
The return type is *proj:sprints*.

tickets

A *proj:tickets* object that contains the tickets associated with the given project.

Returns:
The return type is *proj:tickets*.

13.2.46 proj:sprint

Implements the Storm API for a ProjectSprint

desc

A description of the sprint. This can also be used to set the description.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

name

The name of the sprint. This can also be used to set the name.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

status

The status of the sprint. This can also be used to set the status.

Returns:
The return type may be one of the following: `int`, `null`. When this is used to set the value, it does not have a return type.

tickets

Yields out the tickets associated with the given sprint (no call needed).

Returns:
The return type is `generator`.

13.2.47 proj:sprints

Implements the Storm API for ProjectSprints objects, which are collections of sprints associated with a single project

add(name, period=$lib.null)

Add a sprint.

Args:
name (str): The name for the new ProjectSprint.

period (ival): The time interval the ProjectSprint runs for

Returns:
The newly created *proj:sprint* object The return type is *proj:sprint*.

del(name)

Delete a sprint by name.

Args:
name (str): The name of the Sprint to delete.

Returns:
True if the ProjectSprint can be found and deleted, otherwise False The return type is *boolean*.

get(name)

Get a sprint by name.

Args:
name (str): The name (or iden) of the ProjectSprint to get.

Returns:
The *proj:sprint* object The return type is *proj:sprint*.

13.2.48 proj:ticket

Implements the Storm API for a ProjectTicket.

assignee

The user the ticket is assigned to. This can be used to set the assignee of the ticket.

Returns:
The return type may be one of the following: `int`, `null`. When this is used to set the value, it does not have a return type.

comments

A `proj:comments` object that contains comments associated with the given ticket.

Returns:
The return type is *proj:comments*.

desc

A description of the ticket. This can be used to set the description.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

epic

The epic associated with the ticket. This can be used to set the epic.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

name

The name of the ticket. This can be used to set the name of the ticket.

Returns:
The return type may be one of the following: *str*, `null`. When this is used to set the value, it does not have a return type.

priority

An integer value from the enums [0, 10, 20, 30, 40, 50] of the priority of the ticket. This can be used to set the priority of the ticket.

Returns:
The return type may be one of the following: `int`, `null`. When this is used to set the value, it does not have a return type.

sprint

The sprint the ticket is in. This can be used to set the sprint this ticket is in.

Returns:
The return type may be one of the following: `int`, `null`. When this is used to set the value, it does not have a return type.

status

The status of the ticket. This can be used to set the status of the ticket.

Returns:
The return type may be one of the following: `int`, `null`. When this is used to set the value, it does not have a return type.

13.2.49 proj:tickets

Implements the Storm API for ProjectTickets objects, which are collections of tickets associated with a project

add(name, desc=)

Add a ticket.

Args:
name (str): The name for the new ProjectTicket.

desc (str): A description of the new ticket

Returns:
The newly created *proj:ticket* object The return type is *proj:ticket*.

del(name)

Delete a sprint by name.

Args:
name (str): The name of the ProjectTicket to delete.

Returns:
True if the ProjectTicket can be found and deleted, otherwise False The return type is *boolean*.

get(name)

Get a ticket by name.

Args:
name (str): The name (or iden) of the ProjectTicket to get.

Returns:
The *proj:ticket* object The return type is *proj:ticket*.

13.2.50 queue

A StormLib API instance of a named channel in the Cortex multiqueue.

cull(offs)

Remove items from the queue up to, and including, the offset.

Args:
offs (int): The offset which to cull records from the queue.

Returns:
The return type is `null`.

get(offs=(0), cull=$lib.true, wait=$lib.true)

Get a particular item from the Queue.

Args:
offs (int): The offset to retrieve an item from.

cull (boolean): Culls items up to, but not including, the specified offset.

wait (boolean): Wait for the offset to be available before returning the item.

Returns:
A tuple of the offset and the item from the queue. If wait is false and the offset is not present, null is returned. The return type is *list*.

gets(offs=(0), wait=$lib.true, cull=$lib.false, size=$lib.null)

Get multiple items from the Queue as a iterator.

Args:
offs (int): The offset to retrieve an items from.

wait (boolean): Wait for the offset to be available before returning the item.

cull (boolean): Culls items up to, but not including, the specified offset.

size (int): The maximum number of items to yield

Yields:
Yields tuples of the offset and item. The return type is *list*.

name

The name of the Queue.

Returns:
The type is *str*.

pop(offs=$lib.null, wait=$lib.false)

Pop a item from the Queue at a specific offset.

Args:
offs (int): Offset to pop the item from. If not specified, the first item in the queue will be popped.

wait (boolean): Wait for an item to be available to pop.

Returns:
The offset and item popped from the queue. If there is no item at the offset or the queue is empty and wait is false, it returns null. The return type is *list*.

put(item)

Put an item into the queue.

Args:
item (prim): The item being put into the queue.

Returns:
The return type is `null`.

puts(items)

Put multiple items into the Queue.

Args:
items (list): The items to put into the Queue.

Returns:
The return type is `null`.

size()

Get the number of items in the Queue.

Returns:
The number of items in the Queue. The return type is `int`.

13.2.51 set

Implements the Storm API for a Set object.

add(*items)

Add a item to the set. Each argument is added to the set.

Args:
*items (any): The items to add to the set.

Returns:
The return type is `null`.

adds(*items)

Add the contents of a iterable items to the set.

Args:
*items (any): Iterables items to add to the set.

Returns:
The return type is `null`.

has(item)

Check if a item is a member of the set.

Args:
item (any): The item to check the set for membership.

Returns:
True if the item is in the set, false otherwise. The return type is *boolean*.

list()

Get a list of the current members of the set.

Returns:
A list containing the members of the set. The return type is *list*.

rem(*items)

Remove an item from the set.

Args:
*items (any): Items to be removed from the set.

Returns:
The return type is `null`.

rems(*items)

Remove the contents of a iterable object from the set.

Args:
*items (any): Iterables items to remove from the set.

Returns:
The return type is `null`.

size()

Get the size of the set.

Returns:
The size of the set. The return type is `int`.

13.2.52 spooled:set

A StormLib API instance of a Storm Set object that can fallback to lmdb.

add(*items)

Add a item to the set. Each argument is added to the set.

Args:
*items (any): The items to add to the set.

Returns:
The return type is `null`.

adds(*items)

Add the contents of a iterable items to the set.

Args:
*items (any): Iterables items to add to the set.

Returns:
The return type is `null`.

has(item)

Check if a item is a member of the set.

Args:
item (any): The item to check the set for membership.

Returns:
True if the item is in the set, false otherwise. The return type is *boolean*.

list()

Get a list of the current members of the set.

Returns:
A list containing the members of the set. The return type is *list*.

rem(*items)

Remove an item from the set.

Args:
*items (any): Items to be removed from the set.

Returns:
The return type is `null`.

rems(*items)

Remove the contents of a iterable object from the set.

Args:
*items (any): Iterables items to remove from the set.

Returns:
The return type is `null`.

size()

Get the size of the set.

Returns:
The size of the set. The return type is `int`.

13.2.53 stat:tally

A tally object.

An example of using it:

```
$tally = $lib.stats.tally()

$tally.inc(foo)

for $name, $total in $tally {
    $doStuff($name, $total)
}
```

get(name)

Get the value of a given counter.

Args:
name (str): The name of the counter to get.

Returns:
The value of the counter, or 0 if the counter does not exist. The return type is `int`.

inc(name, valu=(1))

Increment a given counter.

Args:
name (str): The name of the counter to increment.

valu (int): The value to increment the counter by.

Returns:
The return type is `null`.

sorted(byname=$lib.false, reverse=$lib.false)

Get a list of (counter, value) tuples in sorted order.

Args:
byname (bool): Sort by counter name instead of value.

reverse (bool): Sort in descending order instead of ascending order.

Returns:
List of (counter, value) tuples in sorted order. The return type is *list*.

13.2.54 stix:bundle

Implements the Storm API for creating and packing a STIX bundle for v2.1

add(node, stixtype=$lib.null)

Make one or more STIX objects from a node, and add it to the bundle.

Examples:
Example Storm which would be called remotely via the `callStorm()` API:

```
init { $bundle = $lib.stix.bundle() }
#aka.feye.thr.apt1
$bundle.add($node)
fini { return($bundle) }
```

Args:
node (node): The node to make a STIX object from.

stixtype (str): The explicit name of the STIX type to map the node to. This will override the default mapping.

Returns:
The stable STIX id of the added object. The return type is *str*.

pack()

Return the bundle as a STIX JSON object.

Returns:
The return type is *dict*.

size()

Return the number of STIX objects currently in the bundle.

Returns:
The return type is `int`.

13.2.55 storm:query

A storm primitive representing an embedded query.

exec()

Execute the Query in a sub-runtime.

Notes:
The `.exec()` method can return a value if the Storm query contains a `return( ... )` statement in it.

Returns:
A value specified with a return statement, or none. The return type may be one of the following: `null`, `any`.

size(limit=(1000))

Execute the Query in a sub-runtime and return the number of nodes yielded.

Args:
limit (int): Limit the maximum number of nodes produced by the query.

Returns:
The number of nodes yielded by the query. The return type is `int`.

13.2.56 str

Implements the Storm API for a String object.

encode(encoding=utf8)

Encoding a string value to bytes.

Args:
encoding (str): Encoding to use. Defaults to utf8.

Returns:
The encoded string. The return type is *bytes*.

endswith(text)

Check if a string ends with text.

Args:
text (str): The text to check.

Returns:
True if the text ends with the string, false otherwise. The return type is *boolean*.

find(valu)

Find the offset of a given string within another.

Examples:
Find values in the string `asdf`:

```
$x = asdf
$x.find(d) // returns 2
$x.find(v) // returns null
```

Args:
valu (str): The substring to find.

Returns:
The first offset of substring or null. The return type is `int`.

format(**kwargs)

Format a text string from an existing string.

Examples:
Format a string with a fixed argument and a variable:

```
$template='Hello {name}, list is {list}!' $list=(1,2,3,4) $new=$template.
↪format(name='Reader', list=$list)
```

Args:
**kwargs (any): Keyword values which are substituted into the string.

Returns:
The new string. The return type is *str*.

json()

Parse a JSON string and return the deserialized data.

Returns:
The JSON deserialized object. The return type is `prim`.

ljust(size, fillchar=)

Left justify the string.

Args:
size (int): The length of character to left justify.

fillchar (str): The character to use for padding.

Returns:
The left justified string. The return type is *str*.

lower()

Get a lowercased copy of the string.

Examples:
Printing a lowercased string:

```
$foo="Duck"
$lib.print($foo.lower())
```

Returns:
The lowercased string. The return type is *str*.

lstrip(chars=$lib.null)

Remove leading characters from a string.

Examples:
Removing whitespace and specific characters:

```
$strippedFoo = $foo.lstrip()
$strippedBar = $bar.lstrip(w)
```

Args:
chars (str): A list of characters to remove. If not specified, whitespace is stripped.

Returns:
The stripped string. The return type is *str*.

replace(oldv, newv, maxv=$lib.null)

Replace occurrences of a string with a new string, optionally restricting the number of replacements.

Example:
Replace instances of the string “bar” with the string “baz”:

```
$foo.replace('bar', 'baz')
```

Args:
oldv (str): The value to replace.

newv (str): The value to add into the string.

maxv (int): The maximum number of occurrences to replace.

Returns:
The new string with replaced instances. The return type is *str*.

reverse()

Get a reversed copy of the string.

Examples:
Printing a reversed string:

```
$foo="foobar"
$lib.print($foo.reverse())
```

Returns:
The reversed string. The return type is *str*.

rjust(size, fillchar=)

Right justify the string.

Args:
size (int): The length of character to right justify.

fillchar (str): The character to use for padding.

Returns:
The right justified string. The return type is *str*.

rsplit(text, maxsplit=(-1))

Split the string into multiple parts, from the right, based on a separator.

Example:
Split a string on the colon character:

```
($foo, $bar) = $baz.rsplit(":", maxsplit=1)
```

Args:
text (str): The text to split the string up with.

maxsplit (int): The max number of splits.

Returns:
A list of parts representing the split string. The return type is *list*.

rstrip(chars=$lib.null)

Remove trailing characters from a string.

Examples:
Removing whitespace and specific characters:

```
$strippedFoo = $foo.rstrip()
$strippedBar = $bar.rstrip(asdf)
```

Args:
chars (str): A list of characters to remove. If not specified, whitespace is stripped.

Returns:
The stripped string. The return type is *str*.

size()

Return the length of the string.

Returns:
The size of the string. The return type is `int`.

slice(start, end=$lib.null)

Get a substring slice of the string.

Examples:
Slice from index to 1 to 5:

```
$x="foobar"
$y=$x.slice(1,5)  // "ooba"
```

Slice from index 3 to the end of the string:

```
$y=$x.slice(3)  // "bar"
```

Args:
start (int): The starting character index.

end (int): The ending character index. If not specified, slice to the end of the string

Returns:
The slice substring. The return type is *str*.

split(text, maxsplit=(-1))

Split the string into multiple parts based on a separator.

Example:
Split a string on the colon character:

```
($foo, $bar) = $baz.split(":")
```

Args:
text (str): The text to split the string up with.

maxsplit (int): The max number of splits.

Returns:
A list of parts representing the split string. The return type is *list*.

startswith(text)

Check if a string starts with text.

Args:
text (str): The text to check.

Returns:
True if the text starts with the string, false otherwise. The return type is *boolean*.

strip(chars=$lib.null)

Remove leading and trailing characters from a string.

Examples:
Removing whitespace and specific characters:

```
$strippedFoo = $foo.strip()
$strippedBar = $bar.strip(asdf)
```

Args:
chars (str): A list of characters to remove. If not specified, whitespace is stripped.

Returns:
The stripped string. The return type is *str*.

title()

Get a title cased copy of the string.

Examples:
Printing a title cased string:

```
$foo="Hello world."
$lib.print($foo.title())
```

Returns:
The title cased string. The return type is *str*.

upper()

Get a uppercased copy of the string.

Examples:
Printing a uppercased string:

```
$foo="Duck"
$lib.print($foo.upper())
```

Returns:
The uppercased string. The return type is *str*.

13.2.57 telepath:proxy

Implements the Storm API for a Telepath proxy.

These can be created via `$lib.telepath.open()`. Storm Service objects are also Telepath proxy objects.

Methods called off of these objects are executed like regular Telepath RMI calls.

An example of calling a method which returns data:

```
$prox = $lib.telepath.open($url)
$result = $prox.doWork($data)
return ( $result )
```

An example of calling a method which is a generator:

```
$prox = $lib.telepath.open($url)
for $item in $prox.genrStuff($data) {
    $doStuff($item)
}
```

13.2.58 telepath:proxy:genrmethod

Implements the generator methods for the telepath:proxy.

An example of calling a method which is a generator:

```
$prox = $lib.telepath.open($url)
for $item in $prox.genrStuff($data) {
    $doStuff($item)
}
```

13.2.59 telepath:proxy:method

Implements the call methods for the telepath:proxy.

An example of calling a method which returns data:

```
$prox = $lib.telepath.open($url)
$result = $prox.doWork($data)
$doStuff($result)
```

13.2.60 text

A mutable text type for simple text construction.

add(text, **kwargs)

Add text to the Text object.

Args:
text (str): The text to add.

**kwargs (any): Keyword arguments used to format the text.

Returns:
The return type is `null`.

str()

Get the text content as a string.

Returns:
The current string of the text object. The return type is *str*.

13.2.61 trigger

Implements the Storm API for a Trigger.

iden

The Trigger iden.

Returns:
The type is *str*.

move(viewiden)

Modify the Trigger to run in a different View.

Args:
viewiden (str): The iden of the new View for the Trigger to run in.

Returns:
The return type is `null`.

pack()

Get the trigger definition.

Returns:
The definition. The return type is *dict*.

set(name, valu)

Set information in the Trigger.

Args:
name (str): Name of the key to set.

valu (prim): The data to set

Returns:
The return type is `null`.

13.2.62 vault

Implements the Storm API for a Vault.

Callers (instantiation) of this class must have already checked that the user has at least PERM_READ to the vault.

configs

The Vault configs data.

Returns:
The return type is *vault:data*. When this is used to set the value, it does not have a return type.

delete()

Delete the Vault.

Returns:
$lib.true if the vault was deleted, $lib.false otherwise. The return type is *boolean*.

iden

The Vault iden.

Returns:
The type is *str*.

name

The Vault name.

Returns:
The return type is *str*. When this is used to set the value, it does not have a return type.

owner

The Vault owner (user or role iden).

Returns:
The type is *str*.

permissions

The Vault permissions.

Returns:
The return type is *dict*.

scope

The Vault scope.

Returns:
The type is *str*.

secrets

The Vault secrets data.

Returns:
The return type is *vault:data*. When this is used to set the value, it does not have a return type.

setPerm(iden, level)

Set easy permissions on the Vault.

Args:
iden (str): The user or role to modify.

level (str): The easyperm level for the iden. $lib.undef to remove an existing permission.

Returns:
$lib.true if the permission was set, $lib.false otherwise. The return type is *boolean*.

type

The Vault type.

Returns:
The type is *str*.

13.2.63 vault:data

Implements the Storm API for Vault data. This is used for both vault configs and vault secrets.

13.2.64 view

Implements the Storm api for a View instance.

addNode(form, valu, props=$lib.null)

Transactionally add a single node and all it's properties. If any validation fails, no changes are made.

Args:
form (str): The form name.

valu (prim): The primary property value.

props (dict): An optional dictionary of props.

Returns:
The node if the view is the current view, otherwise null. The return type is *node*.

addNodeEdits(edits)

Add NodeEdits to the view.

Args:
edits (list): A list of nodeedits.

Returns:
The return type is `null`.

delMergeRequest()

Remove the existing merge request.

Returns:
The deleted merge request. The return type is *dict*.

delMergeVote(useriden=$lib.null)

Remove a previously created merge vote.

Notes:
The default use case removes a vote cast by the current user. Specifying the useriden parameter allows you to remove a vote cast by another user but requires global admin permissions.

Args:
useriden (str): Delete a merge vote by a different user.

Returns:
The vote record that was removed. The return type is *dict*.

detach()

Detach the view from its parent. WARNING: This cannot be reversed.

Returns:
The return type is `null`.

fork(name=$lib.null)

Fork a View in the Cortex.

Args:
name (str): The name of the new view.

Returns:
The `view` object for the new View. The return type is *view*.

get(name, defv=$lib.null)

Get a view configuration option.

Args:
name (str): Name of the value to get.

defv (prim): The default value returned if the name is not set in the View.

Returns:
The value requested or the default value. The return type is `prim`.

getEdgeVerbs()

Get the Edge verbs which exist in the View.

Yields:
Yields the edge verbs used by Layers which make up the View. The return type is *str*.

getEdges(verb=$lib.null)

Get node information for Edges in the View.

Args:
verb (str): The name of the Edges verb to iterate over.

Yields:
Yields tuples containing the source iden, verb, and destination iden. The return type is *list*.

getFormCounts()

Get the formcounts for the View.

Example:
Get the formcounts for the current View:

```
$counts = $lib.view.get().getFormCounts()
```

Returns:
Dictionary containing form names and the count of the nodes in the View's Layers. The return type is *dict*.

getMergeRequest()

Return the existing merge request or null.

Returns:
The merge request. The return type is *dict*.

getMergeRequestSummary()

Return the merge request, votes, parent quorum definition, and current layer offset.

Returns:
The summary info. The return type is *dict*.

getMerges()

Yields previously successful merges into the view.

Yields:
Yields previously successful merges into the view. The return type is *dict*.

getMergingViews()

Get a list of idens of Views that have open merge requests to this View.

Idens:
The list of View idens that have an open merge request into this View. The return type is *list*.

getPropArrayCount(propname, valu=$lib.undef)

Get the number of individual array property values in the View for the given array property name.

Notes:
This is a fast approximate count calculated by summing the number of array property values in each layer of the view. Property values which are overwritten by different values in higher layers will still be included in the count.

Args:
propname (str): The property name to look up.

valu (any): The value in the array property to look up.

Returns:
The count of nodes. The return type is `int`.

getPropCount(propname, valu=$lib.undef)

Get the number of nodes in the View with a specific property and optional value.

Notes:
This is a fast approximate count calculated by summing the number of nodes with the property value in each layer of the view. Property values which are overwritten by different values in higher layers will still be included in the count.

Args:
propname (str): The property name to look up.

valu (any): The value of the property to look up.

Returns:
The count of nodes. The return type is `int`.

getTagPropCount(tag, propname, form=$lib.null, valu=$lib.undef)

Get the number of nodes in the View with the given tag property and optional value.

Notes:
This is a fast approximate count calculated by summing the number of nodes with the tag property value in each layer of the view. Values which are overwritten by different values in higher layers will still be included in the count.

Args:
tag (str): The tag to look up.

propname (str): The property name to look up.

form (str): The optional form to look up.

valu (any): The value of the property to look up.

Returns:
The count of nodes. The return type is `int`.

iden

The iden of the View.

Returns:
The type is *str*.

layers

The `layer` objects associated with the `view`.

Returns:
The type is *list*.

merge(force=$lib.false)

Merge a forked View back into its parent View.

Args:
force (boolean): Force the view to merge if possible.

Returns:
The return type is `null`.

pack()

Get the View definition.

Returns:
Dictionary containing the View definition. The return type is *dict*.

parent

The parent View. Will be `$lib.null` if the view is not a fork.

Returns:
The type is *str*.

repr()

Get a string representation of the View.

Returns:
A list of lines that can be printed, representing a View. The return type is *list*.

set(name, valu)

Set a view configuration option.

Current runtime updatable view options include:

name (str)
A terse name for the View.

desc (str)
A description of the View.

parent (str)
The parent View iden.

nomerge (bool)
Deprecated - use protected. Updates to this option will be redirected to the protected option (below) until this option is removed.

protected (bool)
Setting to $lib.true will prevent the layer from being merged or deleted.

layers (list(str))
Set the list of layer idens for a non-forked view. Layers are specified in precedence order with the first layer in the list being the write layer.

quorum (dict)
A dictionary of the quorum settings which require users to vote on merges. {

“count”: <int>, “roles”: [<roleid>, ...]

} Once quorum is enabled for a view, any forks must use the setMergeRequest() API to request that the child view is merged. The $view.addMergeVote() API is used for users to add their votes if they have been granted one of the roles listed. Once the number of approvals are met and there are no vetoes, a background process will kick off which merges the nodes and ultimately deletes the view and top layer.

To maintain consistency with the view.fork() semantics, setting the “parent” option on a view has a few limitations:

- The view must not already have a parent
- The view must not have more than 1 layer

Args:
name (str): The name of the value to set.

valu (prim): The value to set.

Returns:
The return type is `null`.

setMergeComment(comment)

Set the main comment/description of a merge request.

Args:
comment (str): The text comment to set for the merge request

Returns:
The updated merge request. The return type is *dict*.

setMergeRequest(comment=$lib.null)

Setup a merge request for the view in the current state.

Args:
comment (str): A text comment to include in the merge request.

Returns:
The newly created merge request. The return type is *dict*.

setMergeVote(approved=$lib.true, comment=$lib.null)

Register a vote for or against the current merge request.

Args:
approved (boolean): Set to (true) to approve the merge or (false) to veto it.

comment (str): A comment attached to the vote.

Returns:
The vote record that was created. The return type is *dict*.

setMergeVoteComment(comment)

Set the comment associated with your vote on a merge request.

Args:
comment (str): The text comment to set for the merge vote

Returns:
The fully updated vote record. The return type is *dict*.

triggers

The `trigger` objects associated with the `view`.

Returns:
The type is *list*.

wipeLayer()

Delete all nodes and nodedata from the write layer. Triggers will be run.

Returns:
The return type is `null`.

13.2.65 xml:element

A Storm object for dealing with elements in an XML tree.

attrs

The element attributes list.

Returns:
The type is *dict*.

find(name, nested=$lib.true)

Find all nested elements with the specified tag name.

Args:
name (str): The name of the XML tag.

nested (bool): Set to $lib.false to only find direct children.

Returns:
A generator which yields xml:elements. The return type is `generator`.

get(name)

Get a single child element by XML tag name.

Args:
name (str): The name of the child XML element tag.

Returns:
The child XML element or $lib.null The return type is *xml:element*.

name

The element tag name.

Returns:
The type is *str*.

text

The element text body.

Returns:
The type is *str*.

CHAPTER

FOURTEEN

SYNAPSE POWER-UPS

Power-Ups are part of The Vertex Project's commercial offering, Synapse Enterprise. Synapse Enterprise is an on-premises solution that includes Optic (the Synapse UI) and all of the Power-Ups. The license includes unlimited users and does not limit the amount of data or number of instances you deploy. We take a white-glove approach to each deployment where we're with you every step of the way from planning deployment sizes to helping to train your analysts.

Feel free to contact us or request a demo instance.

Power-Ups provide specific add-on capabilities to Synapse via Storm Packages and Services. For example, Power-Ups may provide connectivity to external databases, third-party data sources, or enable functionality such as the ability to manage YARA rules, scans, and matches.

For an introduction to Power-Ups from our analysts and seeing them in use, see the following video introducing them:

The Vertex Project is constantly releasing new Power-Ups and expanding features of existing Power-Ups. If you join the `#synapse-releases` channel in Synapse Slack, you can get realtime notices of these updates!

14.1 Rapid Power-Ups

Rapid Power-Ups are delivered to a Cortex as Storm packages directly, without requiring any additional containers to be deployed. This allows users to rapidly expand the power of their Synapse deployments without needing to engage with additional operations teams in their environments. For an introduction to Rapid Power-Ups and some information about publicly available Power-Ups, see the following blog post.

See the Rapid Power-Ups List for a complete list of all available Rapid Power-Ups.

14.1.1 Getting Started with Rapid Power-Ups

Vertex maintains a package repository which allows for loading public and private packages.

If you are a *Synapse User Interface* user, you can navigate to the **Power-Ups Tool** to register your Cortex and configure packages.

Alternatively, one can use the *storm* tool to get started with Rapid Power-Ups in their Cortex.

See our blog article for a step-by step guide to registering your Cortex to install the free `synapse-misp`, `synapse-mitre-attack`, and `synapse-tor` Power-Ups.

14.2 Advanced Power-Ups

Advanced Power-Ups are enhancements to a Cortex which require the deployment of additional containers in order to run their services.

Documentation for specific Advanced Power-Ups can be found here:

- Synapse Backup
- Synapse Fileparser
- Synapse Maxmind
- Synapse GCS
- Synapse Metrics
- Synapse Nettools
- Synapse NSRL
- Synapse Playwright
- Synapse Rapid7
- Synapse Rapid7 SonarRDNS
- Synapse S3
- Synapse Search
- Synapse Sidepocket
- Synapse Swarm
- Synapse Yara
- Synapse Axon Azure

CHAPTER

FIFTEEN

SYNAPSE USER INTERFACE

Optic (the Synapse UI) is part of The Vertex Project's commercial offering, Synapse Enterprise. Synapse Enterprise is an on-premises solution that includes Optic and all of the Power-Ups. The license includes unlimited users and does not limit the amount of data or number of instances you deploy. We take a white-glove approach to each deployment where we're with you every step of the way from planning deployment sizes to helping to train your analysts.

Feel free to contact us or request a demo instance.

For additional information see the Optic Documentation.

CHAPTER

SIXTEEN

SYNAPSE SUPPORT

Information for Vertex support.

16.1 Slack

Best effort chat based support is available through the Synapse Slack. You can find Vertex Project analysts, engineers, and other users who can help with questions you may have.

16.2 Service Desk

Commercial customers have access to the Vertex Support Service Desk. This is the ideal place for customers to submit issues, modeling questions, and feature requests.

CHAPTER

SEVENTEEN

SYNAPSE CHANGELOG

17.1 v2.167.0 - 2024-04-19

17.1.1 Automatic Migrations

- Set the `protected` flag on all Views in the Cortex, using the existing value of the `nomerge` flag. (#3681)
- See *Data Migration* for more information about automatic migrations.

17.1.2 Model Changes

- Updates to the `base` and `file` models. (#3674) (#3688)

  Updated Types

  `file:path`
  Normalizing paths such as `../.././..` previously failed. This now produces an empty path.

  Deprecated Types

  The following types have been marked as deprecated:

  - `edge`
  - `timeedge`

  Deprecated Forms

  The following forms have been marked as deprecated:

  - `graph:cluster`
  - `graph:node`
  - `graph:event`
  - `edge:refs`
  - `edge:has`
  - `edge:wentto`
  - `graph:edge`
  - `graph:timeedge`

17.1.3 Features and Enhancements

- Add `aha.svc.list` and `aha.svc.stat` commands to enumerate the AHA services. Add `$lib.aha` Storm APIs to delete, get, and list the AHA services. (#3685) (#3692) (#3693)
- Add a `protected` option that can be set on Views to prevent merging and deletion. This replaces the `nomerge` option. (#3679)
- Add Beholder events for creating, deleting, and updating Macros. (#3681)
- Update the `StormPkgTest.getTestCore()` API to add a `prepkghook` callback option. This can be used to execute code prior to loading Storm packages. The `getTestCore()` API now waits for `onload` handlers to complete for each package it loads. (#3687)
- Ensure that the `Cell.ahaclient` is fully owned and managed by the `Cell`. It will no longer use a global client that may exist. (#3677)
- Update the `stix2-validator` library constraints to `>=3.2.0,<4.0.0`. Update the allowed range of the `idna` library to `>=3.6,<3.8`. (#3672) (#3684)

17.1.4 Bugfixes

- Asyncio Tasks created by signal handlers on the Base object are now held onto, to ensure that they cannot be garbage collected before or during their task execution. (#3686)
- Update the `Axon.postfiles` and `Axon.wput` APIs to check for the existence of files before attempting to send them over an HTTP connection. (#3682)
- Fix an issue where pruning a non-existent tag mistakenly pruned related tags. (#3673)
- Ensure that macro names are at least 1 character in length. (#3679)
- Fix a bug where `$lib.telepath.open()` could leak Python exceptions into the Storm runtime. (#3685)

17.1.5 Improved Documentation

- Add documentation for `$lib.aha`, `$lib.aha.pool`, and the `aha:pool` type. (#3685)

17.1.6 Deprecations

- Deprecate the use of `hiveboot.yaml` to configure a Cell hive. This will be removed on 2024-05-05. (#3678)
- The `nomerge` option on views has been deprecated. It is automatically redirected to the `protected` option. This redirection will be removed in `v3.0.0`. (#3681)
- The Telepath APIs for interacting with a Cell Hive, `listHiveKey`, `getHiveKeys`, `getHiveKey`, `setHiveKey`, `popHiveKey`, and `saveHiveTree` have been deprecated. The tools `synapse.tools.hive.load` and `synapse.tools.hive.save` have been deprecated. These will be removed in `v3.0.0`. (#3683)
- The `Telepath.Pipeline` class has been marked as deprecated and will be removed in `v3.0.0`. (#3691)

17.2 v2.166.0 - 2024-04-05

17.2.1 Model Changes

- Updates to the `inet`, `ou`, `person` and `risk` models. (#3649) (#3653) (#3657)

  New Forms

  `inet:tls:handshake`
  An instance of a TLS handshake between a server and client.

  `inet:tls:ja3:sample`
  A JA3 sample taken from a client.

  `inet:tls:ja3s:sample`
  A JA3 sample taken from a server.

  `inet:tls:servercert`
  An x509 certificate sent by a server for TLS.

  `inet:tls:clientcert`
  An x509 certificate sent by a client for TLS.

  New Properties

  `risk:extortion`
  The form had the following property added to it:

  `deadline`
  The time that the demand must be met.

  `risk:leak`
  The form had the following properties added on it:

  `extortion`
  The extortion event which used the threat of the leak as leverage.

  `size:bytes`
  The approximate uncompressed size of the total data leaked.

  `it:mitre:attack:technique`
  The form had the following properties updated on it:

  `name`
  This property is now lower-cased and single spaced.

  Deprecated Forms

  The following forms have been marked as deprecated:

  `inet:ssl:cert`
  Please use `inet:tls:clientcert` or `inet:tls:servercert`.

  Column Display Hints

  The following forms had column display hints added to them:

  `ou:campaign` `ou:conference` `ou:goal` `ou:org` `ou:team` `ou:technique` `ps:contact` `ps:skill` `ps:proficiency` `risk:threat` `risk:compromise` `risk:mitigation` `risk:tool:software`

  Light Edges

uses

When used with a `risk:extortion` and an `ou:technique` node, the edge indicates the attacker used the technique to extort the victim.

17.2.2 Features and Enhancements

- When setting a tag on a node, the tag value is now redirected based on parent tags having `:isnow` properties set. (#3650)
- Add a `$lib.spooled.set()` Storm API. This can be used to get a `spooled:set` object. This set will offload the storage of its members to a temporary location on disk when it grows above a certain size. (#3632)
- Add a `$lib.cache.fixed()` Storm API. This can be used to get a `cache:fixed` object. This cache will execute user provided callbacks written in Storm upon a cache miss. (#3661)
- Add a `pool` option to Cron jobs. This can be set to True to enable a Cron job storm query to be executed on a Storm pool member. (#3652)
- Add a `pool` option to Extended HTTP API handlers. This can be set to True to enable an HTTP request handler to be executed on a Storm pool member. (#3663) (#3667)
- Add a new Storm API, `$lib.cortex.httpapi.getByPath()`, that can be used to get an `http:api` object by its path. The `path` value is evaluated in the same order that the HTTP endpoint resolves the handlers. (#3663)
- Add `--list` and `--gate` options to `synapse.tools.modrole` and `synapse.tools.moduser`. (#3632)
- Add a `view.getMergingViews()` Storm API. This returns a list of view idens that have open merge requests on a view. (#3666)
- The Storm API `show:storage` option now includes storage information for any embedded properties. (#3656)
- Update the `LinkShutDown` exception that a Telepath client may raise to indicate that the connection has been disconnected. (#3640)
- Add repr functions for printing the `aha:pool` and `http:api` objects in Storm. (#3663) (#3665)
- The Telepath `Pool` object has been replaced with a new object, `ClientV2`. This is now the only object returned by the `synapse.telepath.open()` API. This is an AHA pool aware Client which can be used to connect to an AHA pool. (#3662)
- Remove the unused Provenance subsystem from the Cortex. (#3655)
- Constrain the `stix2-validator` library to `3.0.0,<3.2.0` due to an API change. This constraint is expected be changed in the next release. (#3669)

17.2.3 Bugfixes

- Fix a bug where a Cortex `promote()` call could hang when tearing down any running Cron jobs. Cron jobs cancelled during a promotion event will be logged but their cancelled status will not be recorded in the Nexus. (#3658)
- Fix a bug where the Storm pool configuration could cause a Cortex to fail to start up. The Storm pool is now configured upon startup but its use is blocked until the Storm pool is ready to service requests. (#3662)
- Ensure that the URL argument provided to `cortex.storm.pool.set` can be parsed as a Telepath URL. Previously any string input was accepted. (#3665)

17.2.4 Improved Documentation

- Update the list of Cortex permissions in the Admin Guide to include `service.add`, `service.del`, `service.get`, and `service.list`. (#3647)
- Update the docstring for the Storm `cortex.storm.pool.del` command to note the effects of removing a pool and the interruption of running queries. (#3665)
- Update the documentation for the Storm `http:api` object to include the `methods` attribute. (#3663)

17.2.5 Deprecations

- The Telepath `task:init` message format has been marked as deprecated and will be removed in `v3.0.0`. This should not affect any users using Synapse `v2.x.x` in their client code. (#3640)
- The authgate with the name `cortex` is not used for permission checking and will be removed in `v3.0.0`. At startup, the Cortex will now check for any use of this authgate and log warning messages. Attempts to set permissions with this gateiden via Storm will produce `warn` messages. (#3648)

17.3 v2.165.0 - 2024-03-25

17.3.1 Automatic Migrations

- Re-normalize `risk:mitigation:name`, `it:mitre:attack:technique:name`, and `it:mitre:attack:mitigation:name` secondary properties. (#3585)
- Re-normalize `velocity` properties which are float values. (#3616)
- See *Data Migration* for more information about automatic migrations.

17.3.2 Model Changes

- Add a new model, `sci`, for modeling elements of the scientific method. Updates to the `econ`, `file`, `infotech`, `inet`, `ou`, `ps`, and `risk` models. (#3559) (#3585) (#3595) (#3604) (#3606) (#3622) (#3635)

  New Forms

  `econ:acct:receipt`
  : A receipt issued as proof of payment.

  `econ:acct:invoice`
  : An invoice issued requesting payment.

  `econ:bank:account:type:taxonomy`
  : A bank account type taxonomy.

  `econ:bank:account`
  : A bank account.

  `econ:bank:balance`
  : A balance contained by a bank account at a point in time.

  `econ:bank:statement`
  : A statement of bank account payment activity over a period of time.

  `econ:bank:aba:rtn`
  : An American Bank Association (ABA) routing transit number (RTN).

`econ:bank:iban`
An International Bank Account Number.

`econ:bank:swift:bic`
A Society for Worldwide Interbank Financial Telecommunication (SWIFT) Business Identifier Code (BIC).

`risk:vulnerable`
Indicates that a node is susceptible to a vulnerability.

`sci:hypothesis:type:taxonomy`
A taxonomy of hypothesis types.

`sci:hypothesis`
A hypothesis or theory.

`sci:experiment:type:taxonomy`
A taxonomy of experiment types.

`sci:experiment`
An instance of running an experiment.

`sci:observation`
An observation which may have resulted from an experiment.

`sci:evidence`
An assessment of how an observation supports or refutes a hypothesis.

Updated Properties

`risk:mitigation`
The form had the following properties updated on it:

`name`
This property is now lower-cased and single spaced.

`it:mitre:attack:technique`
The form had the following properties updated on it:

`name`
This property is now lower-cased and single spaced.

`it:mitre:attack:mitigation`
The form had the following properties updated on it:

`name`
This property is now lower-cased and single spaced.

New Properties

`econ:acct:payment`
The form had the following properties added to it:

`from:account`
The bank account which made the payment.

`to:account`
The bank account which received the payment.

`invoice`
The invoice that the payment applies to.

`receipt`
The receipt that was issued for the payment.

`file:mime:image`
The interface had the following property added to it:

`text`
The text contained within the image.

`inet:email:message`
The form had the following property added to it:

`flow`
The inet:flow which delivered the message.

`ou:id:number`
The form had the following property added to it:

`issuer`
The contact information of the office which issued the ID number.

`risk:threat`
The form had the following property added to it:

`mitre:attack:group`
A mapping to a MITRE ATT&CK group if applicable.

`risk:tool:software`
The form had the following property added to it:

`mitre:attack:software`
A mapping to a MITRE ATT&CK software if applicable.

`risk:mitigation`
The form had the following property added to it:

`mitre:attack:mitigation`
A mapping to a MITRE ATT&CK mitigation if applicable.

Deprecated Forms

The following forms have been marked as deprecated:

`risk:hasvuln`
Please use `risk:vulnerable`.

Light Edges

`has`
When used with an `econ:bank:statement` and an `econ:acct:payment`, the edge indicates the bank statement includes the payment.

When used with an `ou:org` node, the edge indicates the organization is or was in possession of the target node.

When used with a `ps:contact` node, the edge indicates the contact is or was in possession of the target node.

When used with a `ps:person` node, the edge indicates the person is or was in possession of the target node.

When used with a `sci:observation` node, the edge indicates the observations are summarized from the target nodes.

When used with an `sci:evidence` node, the edge indicates the evidence includes observations from the target nodes.

`owns`
When used with an `ou:org` node, the edge indicates the organization owns or owned the target node.

When used with a `ps:contact` node, the edge indicates the contact owns or owned the target node.

When used with a `ps:person` node, the edge indicates the person owns or owned the target node.

uses

When used with a `sci:experiment` node, the edge indicates the experiment used the target nodes when it was run.

17.3.3 Features and Enhancements

- Change the compression mode used when streaming Cell backups to speed up the backup process. (#3608)
- When a Cell is mirroring, gracefully go into read-only mode if the leader is a greater version than the mirror. (#3581) (#3631)
- Add `null` as a constant that can be used in Storm expression syntax. (#3600)
- Add `cortex.storm.pool.get`, `cortex.storm.pool.set`, and `cortex.storm.pool.del` commands to manage the Storm query pool which may be used by the Cortex. This replaces the experimental support added in `v2.160.0` for Storm query pool configuration. The experimental Cortex configurations options `storm:pool`, `storm:pool:timeout:sync`, and `storm:pool:timeout:connection` have been removed. (#3602)
- Add `$lib.regex.escape()` API for escaping strings which may be used as regular expression patterns. (#3605)
- Add `View.setMergeComment()` and `View.setMergeVoteComment()` Storm APIs for setting comments on merge requests and merge votes. (#3597)
- Add handlers to the `float`, `int`, and `str` types to handle norming Storm `Number` objects. (#3601)
- Add a new Storm command, `gen.geo.place`, to generate a `geo:place` node by name. (#3620)
- Add an optional reporter name argument to the Storm command `gen.risk.vuln`. (#3628)
- Add a `norm` option to the `$node.difftags()` command. (#3612)
- Add logging around the leader promotion and handoff actions. (#3615)
- Add Telepath APIs to AHA for clearing unused provisioning information. (#3607)

17.3.4 Bugfixes

- Fix a bug where Cortex Cron jobs could start prior to data migrations having completed running. (#3610)
- Fix an issue where `node.prop.set` and `node.prop.del` permissions were not being properly checked. (#3627)
- Fix a bug in the Storm `merge` command where the destination layer was not being properly checked for property set and deletion permissions. (#3627)
- Fix a bug in the Storm `copyto` command where the destination layer was not being properly checked for property set permissions. (#3641)
- Fix an error when granting a role admin permissions on a vault. (#3603)
- Prevent the `synapse.tools.easycert` tool from making certificates with names greater than 64 characters in length. Prevent AHA provisioning from creating provisioning requests which would exceed that length. (#3609)
- Fix an issue with the `velocity` base type returning a float instead of an integer when handling a string value without a unit. (#3616)
- Fix an issue that could occur when pivoting from a secondary property to a form when using variables for the source and target values. (#3618)

- Fix a syntax parsing issue when using the try-set-plus or try-set-minus operator to update an array property on a node using a variable for the property name. (#3630)
- Fix an issue with AHA service pools where their Telepath Clients were not configured for use as `aha://` clients. (#3643)
- Fix an issue with AHA service pools where a fini'd Proxy was not properly cleaned up. (#3645)

17.3.5 Improved Documentation

- Update Storm pivot documentation to add additional examples. (#3599)
- Update the Cortex deployment guide to include a step to configure a Storm query pool. (#3602)

17.3.6 Deprecations

- The tool `synapse.tools.cellauth` has been marked as deprecated and will be removed in `v3.0.0`. (#3587)
- The tool `synapse.tools.cmdr` has been marked as deprecated and will be removed in `v3.0.0`. (#3589)
- The Storm `$lib.model.edge` APIs have been marked as deprecated and will be removed in `v3.0.0`. (#3623)
- The `CoreAPI.enableMigrationMode()` and `CoreAPI.disableMigrationMode()` Telepath methods have been marked as deprecated and will be removed after 2024-05-05. (#3610)
- The Cortex configuration options `cron:enable` and `trigger:enable` have been marked as deprecated and will be removed in `v3.0.0`. These configuration options no longer control cron or trigger behavior. (#3610)
- The Storm Package `synapse_minversion` key has been deprecated and will be removed in `v3.0.0`. Package authors should use the `synapse_version` key to specify a version range for Synapse they support. An example is the string `>=2.165.0,<3.0.0`. (#3593)

17.4 v2.164.0 - 2024-03-01

17.4.1 Features and Enhancements

- Update the Beholder messages `view:merge:init`, `view:merge:prog`, and `view:merge:fini` to add `merge` and `vote` information. (#3580)
- When optimizing Storm lift operations, skip lifts that would be fully filtered out. (#3582)
- Add `tmpdir` information to the `getSystemInfo()` APIs. This is the directory that the service would use for creating any temporary files. (#3583)
- Update the `synapse.tools.modrole` tool to add a `--del` option to delete a role. (#3586)
- Add the `reporter` `ou:org` to `ou:campaign` nodes generated with `gen.ou.campaign` (#3594)
- The `synapse.lib.certdir.CertDir` class has been updated to use the `cryptography` APIs instead of the `PyOpenSSL` APIs where possible. The `CertDir` APIs no longer return `PyOpenSSL` objects, and now return `cryptography` related objects. (#3568)
- Update the `cryptography` and `PyOpenSSL` libraries to require their latest versions. (#3568)

17.4.2 Bugfixes

- Model interfaces now populate properties for the sub-interfaces. (#3582)
- Use `tostr` on property and form names when computing lifts and pivots to avoid a Python `AttributeError` exception. Invalid types will now raise a `StormRuntimeException`. (#3584)

17.4.3 Deprecations

- The tool `synapse.tools.cellauth` has been marked as deprecated and will be removed in `v3.0.0`. (#3587)
- The tool `synapse.tools.cmdr` has been marked as deprecated and will be removed in `v3.0.0`. (#3589)

17.5 v2.163.0 - 2024-02-21

17.5.1 Features and Enhancements

- Add Storm API methods to `$lib.axon` which share the functionality of `$lib.bytes` APIs. These include `$lib.axon.has`, `$lib.axon.hashset`, `$lib.axon.put`, `$lib.axon.size`, and `$lib.axon.upload`. (#3570) (#3576)
- Add support for user provided certificates for doing mTLS in Storm HTTP requests. (#3566)
- Enable constructing a guid in Storm from a single value with `$lib.guid(valu=$item)`. (#3575)

17.6 v2.162.0 - 2024-02-15

17.6.1 Model Changes

- Updates to the `inet`, `infotech`, `ou`, `proj`, and `risk` models. (#3549) (#3551) (#3564)

  New Properties

  `inet:email:message`
  : The form had the following properties added to it:

    `received:from:ipv4`
    : The sending SMTP server IPv4, potentially from the Received: header.

    `received:from:ipv6`
    : The sending SMTP server IPv6, potentially from the Received: header.

    `received:from:fqdn`
    : The sending server FQDN, potentially from the Received: header.

  `ou:oid:type`
  : The form had the following property added to it:

    `url`
    : The official URL of the issuer.

  `proj:project`
  : The form had the following property added to it:

    `type`
    : The project type.

`risk:alert`
The form had the following properties added to it:

`status`
The status of the alert.

`assignee`
The Synapse user who is assigned to investigate the alert.

`ext:assignee`
The alert assignee contact information from an external system.

`risk:mitigation`
The form had the following properties added to it:

`reporter`
The organization reporting on the mitigation.

`reporter:name`
The name of the organization reporting on the mitigation.

`tag`
The tag used to annotate nodes which have the mitigation in place.

New Forms

`proj:project:type:taxonomy`
A type taxonomy for projects.

Deprecated Properties

`it:mitre:attack:group`
The `it:mitre:attack:group` form had the following property marked as deprecated:

- `tag`

`it:mitre:attack:tactic`
The `it:mitre:attack:tactic` form had the following property marked as deprecated:

- `tag`

`it:mitre:attack:technique`
The `it:mitre:attack:technique` form had the following property marked as deprecated:

- `tag`

`it:mitre:attack:software`
The `it:mitre:attack:software` form had the following property marked as deprecated:

- `tag`

`it:mitre:attack:campaign`
The `it:mitre:attack:campaign` form had the following property marked as deprecated:

- `tag`

17.6.2 Features and Enhancements

- Add Storm API methods for inspecting and manipulating dictionary objects in Storm. These are `$lib.dict.has()`, `$lib.dict.keys()`, `$lib.dict.pop()`, `$lib.dict.update()`, and ``$lib.dict.values()` (#3548)
- Add a `json()` method to the `str` type in Storm to deserialize a string as JSON data. (#3555)
- Add an `_ahainfo` attribute to the `Telepath.Proxy`, containing AHA service name information if that is provided to the Dmon. (#3552)
- Add permissions checks to `$lib.bytes` APIs using `axon.has` for APIs that check for information about the Axon or metrics; and `axon.upload` for APIs which put bytes in the Axon. These are checked with `default=True` for backward compatibility. (#3563)
- The rstorm `storm-svc` and `storm-pkg` directives now wait for any `onload` handlers to complete. (#3567)
- Update the Synapse Python package trove classifiers to list the platforms we support using Synapse with. (#3557)

17.6.3 Bugfixes

- Fix a bug in the `Cell.updateHttpSessInfo()` API when the Cell does not have the session in memory. (#3556)
- Fix a bug where a user was allowed to vote for their own View merge request. (#3565)
- Include Storm variables from the current and parent scopes when resolving STIX properties and relationships. (#3571)

17.6.4 Improved Documentation

- Update the Storm automation documentation. Added additional information about permissions used to manage automations. Added examples for `edge:add` and `edge:del` triggers. Added examples for managing Macro permissions. (#3547)
- Update the Storm filtering and lifting documentation to add information about using interfaces and wildcard values with those operations. (#3560)
- Update the Synapse introduction to note that Synapse is not intended to replace big-data or data-lake solutions. (#3553)

17.6.5 Deprecations

- The Storm function `$lib.dict()` has been deprecated, in favor of using the `({"key": "value"})` style syntax for directly declaring a dictionary in Storm. (#3548)
- Writeback layer mirrors and upstream layer mirrors have been marked as deprecated configuration options. (#3562)

17.7 v2.161.0 - 2024-02-06

17.7.1 Features and Enhancements

- Add a Storm command `gen.it.av.scan.result` to help generate `it:av:scan:result` nodes. (#3516)
- Add item specific error message when users do not have sufficient permissions on an object which is using easyperms. (#3532)
- Ensure that Nexus events which are written to the log are always applied and cannot be cancelled while the Nexus handler is running. (#3518)
- Add `getMergeRequest()` and `getMergeRequestSummary()` Storm APIs to the `View` object, in order to get information about View merges via Storm. (#3541)
- Add AHA information to the output of the `Cell.getCellInfo()` API. This includes the service name, leader, and network. (#3519)
- Logs related to AHA service registration and setting services as offline are now logged at the `INFO` level. (#3534)
- When creating Cron jobs and Triggers, record their creation time. (#3521) (#3538)
- Add a `Cell.updateHttpSessInfo()` API to set multiple keys at once on a HTTP session. (#3544)
- Update the allowed versions of the `cbor2` and `` pycryptodome`` libraries. (#3540)

17.7.2 Bugfixes

- The Storm API for creating websockets, `$lib.inet.http.connect()`, did not properly handle the `ssl_verify` argument, causing SSL verification of Websocket requests to default to being disabled. This argument is now handled correctly, with SSL verification being enabled by default. (#3527)
- Fix a bug in embedded Storm queries where they failed to grab their variables properly. (#3531)
- Fix a bad variable reference in the Storm `graph` implementation. (#3531)
- Fix a bug where modifying nodes in a Storm Dmon did not properly update the in-flight node. (#3520)

17.7.3 Improved Documentation

- Update the Cortex admin guide with additional information about removing extended forms and properties. (#3510)
- Update the Data Model documentation to include additional information about extended forms and properties. (#3523)
- Update the Data Model documentation to include information about property interfaces. (#3523)

17.8 v2.160.0 - 2024-01-24

17.8.1 Automatic Migrations

- Update `inet:ipv6` nodes to set their `:type` and `:scope` properties. (#3498)
- Update existing layer push and layer pull configurations to set the default chunk size and queue size values on them. (#3480)
- See *Data Migration* for more information about automatic migrations.

17.8.2 Model Changes

- Updates to the `infotech`, `ou`, and `risk` models. (#3501) (#3504) (#3498)

  New Properties

  `risk:vuln`
  The form had the following properties added to it:

  `severity`
  The severity of the vulnerability.

  `priority`
  The priority of the vulnerability.

  `inet:ipv6`
  The form had the following properties added to it:

  `type`
  The type of IP address (e.g., private, multicast, etc.).

  `scope`
  The IPv6 scope of the address (e.g., global, link-local, etc.).

  Updated Types

  `it:exec:proc`
  This now inherits the `it:host:activity` interface.

  `it:exec:thread`
  This now inherits the `it:host:activity` interface.

  `it:exec:loadlib`
  This now inherits the `it:host:activity` interface.

  `it:exec:mmap`
  This now inherits the `it:host:activity` interface.

  `it:exec:mutex`
  This now inherits the `it:host:activity` interface.

  `it:exec:pipe`
  This now inherits the `it:host:activity` interface.

  `it:exec:url`
  This now inherits the `it:host:activity` interface.

  `it:exec:bind`
  This now inherits the `it:host:activity` interface.

`it:exec:file:add`
: This now inherits the `it:host:activity` interface.

`it:exec:file:read`
: This now inherits the `it:host:activity` interface.

`it:exec:file:write`
: This now inherits the `it:host:activity` interface.

`it:exec:file:del`
: This now inherits the `it:host:activity` interface.

`it:exec:reg:get`
: This now inherits the `it:host:activity` interface.

`it:exec:reg:set`
: This now inherits the `it:host:activity` interface.

`it:exec:reg:del`
: This now inherits the `it:host:activity` interface.

17.8.3 Features and Enhancements

- Add tab completion of commands, forms, properties, tags, and `$lib.` functions the Storm CLI tool. (#3493) (#3507)
- Add `node.set.<form>.<prop>` and `node.del.<form>.<prop>` permissions conventions to the Cortex for property sets and deletes. (#3505)
- Add experimental support for Storm query offloading to the Cortex. This can be used to offload Storm queries to an AHA service pool. This can be configured with the `storm:pool` option on the Cortex. (#3452) (#3513)
- Add a `--deledges` option to the `delnode` command. This deletes the N2 edges for a node before deleting the node. (#3503)
- When creating layer push or pull configurations, the chunk size and queue size can now be set. (#3480)
- Add a `cell.hasHttpSess()` API to check if a given Cell has a known HTTP session. (#3485)
- Fire a `core:pkg:onload:complete` event when a Storm package `onload` handler is completed. This can be used when writing unit tests for Rapid Power-ups. (#3497)

17.8.4 Bugfixes

- Remove dataname index entries when removing all nodedata from a node. (#3499)
- Fix an issue with `tagprops` not being correctly returned in `$node.getByLayer()`. (#3500)
- Fix an issue with the `edges.del` command when using the `--n2` option. This now behaves correctly when the N1 node does not exist. (#3506)
- Fix an issue with duplicate properties being tracked in the property type map of the data model. This could have resulted in multiple nodes being lifted with interface properties. (#3512)

17.8.5 Improved Documentation

- Update Storm filter documentation. Additional information about tag globbing and interval filtering has been included. (#3489)

17.9 v2.159.0 - 2024-01-16

17.9.1 Automatic Migrations

- Update any extended model elements which used the `taxonomy` interface to now use the `meta:taxonomy` interface. (#3334)
- See *Data Migration* for more information about automatic migrations.

17.9.2 Features and Enhancements

- Add support for lifting, pivoting, and filtering using wildcards, lists, variables, and interfaces as form and property names. (#3334)
- Migrate the name of the `taxonomy` interface to `meta:taxonomy`. (#3334)
- Update the pinned version of the `lark` library to `1.1.9` for compatibility with Python 3.11.7. (#3488)

17.9.3 Bugfixes

- Prevent re-adding extended model elements in Nexus handlers. (#3486)
- Add missing permissions checks on the `$lib.axon.urlfile()` API. This now requires the `node.add.file:bytes` and `node.add.inet:urlfile` permissions. (#3490)
- Fix the permission checking for Vaults to check the Storm runtime `asroot` status. (#3492)
- Fix an issue with `$lib.stix.import.ingest()` not converting `bundle` to a dictionary. (#3495)

17.9.4 Improved Documentation

- Add documentation for the `reverse` keyword. (#3487)
- Clarify the use of the "try" operator (`+?`) in edit operations. (#3482) (#3487)
- Update Storm lift documentation to add additional examples and clarify existing documentation. (#3487)
- Update Storm data modification documentation to add additional examples and clarify existing documentation. (#3482)

17.10 v2.158.0 - 2024-01-03

17.10.1 Features and Enhancements

- Update the allowed versions of the``fastjsonschema``, `idna`, `pygments`, and `aiosmtplib` libraries. (#3478)

17.10.2 Bugfixes

- Fix a bug where the `role:add` and `user:add` Nexus handlers could raise an exception when being called by a service mirror. (#3483)

17.10.3 Improved Documentation

- Update the Storm command reference guide. (#3481)
- Update the Synapse glossary. (#3481)

17.11 v2.157.0 - 2023-12-21

17.11.1 Features and Enhancements

- Added vaults feature for storing and sharing secret values (such as API keys) and associated configuration settings. Vaults can be shared with and used by another user without them being able to see the enclosed secret values. (#3319) (#3461)
- Added Storm commands to interact with vaults: `vaults.*`. (#3319)
- Added Storm library to interact with vaults: `$lib.vaults.*`. (#3319)
- Add merge request voting and history tracking for full View merges. (#3466) (#3473) (#3475)
- Add service pooling support to AHA. This allows for dynamic service topologies and distributed Telepath API calls. (#3353) (#3477)
- Add user managed API keys that can be used to access HTTP API endpoints. (#3470)
- Added an `--optsfile` option to the Storm CLI tool. This can be used to specify opts to the CLI tool via YAML. See *Storm Opts* for details about available options. (#3468)
- Cron status changes are now persisted through the Nexus. (#3460)
- Add a `show:storage` option to the Storm runtime opts to include the storage node data in the `node` message. (#3471)

17.11.2 Bugfixes

- Log a warning message when calling the Python `User.pack(packroles=True)` method when a user role is missing from the Auth subsystem. A missing role previously caused an `AttributeError` exception. (#3469)
- Ensure the Nexus `view:detach` event is idempotent. (#3474)
- Fix an issue where Storm subqueries containing non-runtsafe values could potentially not execute. (#3443)

17.12 v2.156.0 - 2023-12-08

17.12.1 Model Changes

- Updates to the `infotech`, `ou`, and `risk` models. (#3436) (#3438) (#3446) (#3447)

  New Properties

  `it:av:scan:result`
  The form had the following properties added to it:

  `target:ipv4`
  The IPv4 address that was scanned to produce the result.

  `target:ipv6`
  The IPv6 address that was scanned to produce the result.

  `ou:campaign`
  The form had the following property added to it:

  `mitre:attack:campaign`
  A mapping to a Mitre ATT&CK campaign if applicable.

  `risk:vuln`
  The form had the following property added to it:

  `id`
  An identifier for the vulnerability.

  New Forms

  `it:mitre:attack:campaign`
  A Mitre ATT&CK Campaign ID.

  `risk:technique:masquerade`
  Represents the assessment that a node is designed to resemble another in order to mislead.

  Updated Types

  `it:os:windows:sid`
  The regular expression used to validate the SID has been updated to allow modeling well-known SID values.

17.12.2 Features and Enhancements

- Add an `empty` keyword to Storm to conditionally execute queries when there are no nodes in the pipeline. (#3434)
- Add Storm APIs for getting property counts for a given `layer` or `view`.. These APIs are `getPropCount()`, `getPropArrayCount()`, `getTagPropCount()`. (#3435)
- Add a new permission, `view.fork`, which can be used to control access for forking a view. This permission defaults to being allowed. (#3437)
- Add Storm operators to allow pivoting and joining across light edges. The following examples show pivoting across `refs` edges and joining the destination nodes with the inbound nodes: `-(refs)+>` and `<+(refs)-`. (#3441)
- Add Storm operators to do pivot out and join (`--+>`) and pivot in and join (`<+--`) operations across light edges. (#3441) (#3442)
- Storm subqueries used to assign a value now always run. (#3445)
- Non-runtsafe `try...catch` blocks in Storm now run when there are no inbound nodes. (#3445)
- The Storm API `$lib.storm.eval()` now logs its `text` argument to the `synapse.storm` logger. (#3448)
- Add a `--by-name` argument to the Storm `stats.countby` command. This can be used to sort the results by name instead of count. (#3450)
- Add a new Storm API `$lib.gis.bbox()` to allow computing geospatial bounding boxes. (#3455)

17.12.3 Bugfixes

- Prevent recursion errors in `inet:fqdn` onset handlers. (#3433)
- When dereferencing a list or dictionary object off of a Node in Storm, the returned value is now a copy of the value. This avoids the situation where modifying the deferenced value appeared to alter the node but did not actually result in any edits to the underlying data. (#3439)
- Add a missing sub-query example to Storm `for` loop documentation. (#3451)
- Fix an issue where attempting to norm an IPv4 with an invalid netmask would raise a Python error. (#3459)

17.12.4 Deprecations

- Deprecated Cortex and splice related APIs which were marked for removal after 2023-10-01 have been removed. The list of these APIs can be found at *API Deprecation Notice - 2023-10-01*. These additional splice related changes have also been made:

  The HTTP API `/api/v1/storm` now sets the default `editformat` opt value to `nodeedits`. Previously this API produced splice changes by default.

  The `synapse.tools.cmdr storm` command no longer displays splices.

  The `synapse.tools.cmdr log` command no longer records splices.

  The `synapse.tools.csvtool` tool no longer records or displays splices.

  The `synapse.tools.feed` tool no longer supports splices or nodeedits as input and the splice documentation example has been removed.

  (#3449)
- The deprecated function `synapse.common.aclosing()` has been removed. (#3449)

- Provisioning a Synapse service with AHA now always updates the local CA certificate and generates new host and user certificates for the service. Previously these would not be regenerated if the CA or service names did not change. (#3457)

17.13 v2.155.0 - 2023-11-17

17.13.1 Model Changes

- Updates to the `infotech`, `proj`, and `risk` models. (#3422)

  New Properties

  `proj:ticket`
  : The form had the following property added to it:

    `ext:assignee`
    : Ticket assignee contact information from an external system.

  `risk:alert`
  : The form had the following property added to it:

    `severity`
    : A severity rank for the alert.

  `it:exec:query`
  : The form had the following property added to it:

    `offset`
    : The offset of the last record consumed from the query.

  New Forms

  `it:av:scan:result`
  : The result of running an antivirus scanner.

  Updated Properties

  `risk:alert`
  : The form had the following properties updated on it:

    `priority`
    : The type of this property has been changed from an `int` to `meta:priority`.

  `risk:attack`
  : The form had the following properties updated on it:

    `severity`
    : The type of this property has been changed from an `int` to `meta:severity`.

  `risk:compromise`
  : The form had the following properties updated on it:

    `severity`
    : The type of this property has been changed from an `int` to `meta:severity`.

  Deprecated Forms

  The following forms have been marked as deprecated:

  `it:av:sig`
  : Please use `it:av:scan:result`.

it:av:filehit
Please use `it:av:scan:result`.

it:av:prochit
Please use `it:av:scan:result`.

17.13.2 Features and Enhancements

- Add a `detach()` method to the Storm `view` object. This will detach a forked View from its parent. (#3423)
- Change the method used to generate the `took` value in the Storm `fini` message to use a monotonic clock. (#3425)
- Performing an invalid "pivot in" operation with a form target (`<- some:form`) now raises a `StormRuntimeError` instead of silently doing nothing. (#3426)
- Allow relative properties on the right hand side of a filter operation when using Storm expression syntax. (#3424)
- Add an `/api/v1/logout` method on the Cell to allow HTTPS users to logout of their sessions. (#3430)
- Allow taxonomy prefix lift and filter operations to work with taxon parts. (#3429)
- Update the allowed versions of the `cbor2`, `pycryptodome`, `pygments`, `vcrpy`, and `xxhash` libraries. Update the pinned version of the `lark` library. (#3418)

17.13.3 Bugfixes

- Fix a performance regression in graph projection for computing large graphs in Storm. (#3375)
- Fix a conflict between Storm `$lib.inet.http` functions and `vcrpy` where `json` and `data` args shouldn't be passed together. (#3428)

17.13.4 Improved Documentation

- Fix an error in the Cortex mirror deployment guide. The example `docker-compose.yaml` was missing the environment variables for `SYN_CORTEX_AXON` and `SYN_CORTEX_JSONSTOR`. (#3430)

17.14 v2.154.1 - 2023-11-15

This release is for updating the version of the `cryptography` package in Synapse containers to `41.0.5`.

17.15 v2.154.0 - 2023-11-15

17.15.1 Automatic Migrations

- Update the `inet:ipv4:type` value for RFC6598 addresses to `shared`. (#3410)
- See *Data Migration* for more information about automatic migrations.

17.15.2 Model Changes

- Update to the `inet` and `ou` models.

  (#3406) (#3407) (#3410) (#3416)

  Updated Types

  `inet:ipv4`
  : RFC6598 addresses now have a `:type` property value of `shared`.

  `inet:url`
  : Accept Microsoft URLPrefix strings with a strong wildcard host value.

    Add a check to prevent creating `inet:url` nodes with an empty host and path part, such as `inet:url=file://''`.

  New Properties

  `ou:org`
  : The form had the following property added to it:

    `tag`
    : A base tag used to encode assessments made by the organization.

  `risk:compromise`
  : The form had the following properties added to it:

    `ext:id`
    : An external unique ID for the compromise.

    `url`
    : A URL which documents the compromise.

  `risk:alert`
  : The form had the following property added to it:

    `host`
    : The host which generated the alert.

  New Forms

  `ou:requirement`
  : A specific requirement.

  `risk:leak`
  : An event where information was disclosed without permission.

  `risk:leak:type:taxonomy`
  : A taxonomy of leak event types

  `risk:extortion`
  : An event where an attacker attempted to extort a victim.

  `risk:extortion:type:taxonomy`
  : A taxonomy of extortion event types.

  Light Edges

  `leaked`
  : When used with a `risk:leak` node, the edge indicates the leak included the disclosure of the target node.

  `leveraged`
  : When used with a `risk:extortion` node, the edge indicates the extortion event was based on attacker access to the target node.

meets
When used with a `ou:requirement` node, the edge indicates the requirement was met by the source node.

17.15.3 Features and Enhancements

- Add `edge:add` and `edge:del` as trigger conditions. These trigger when light edges are added or removed from a node. (#3389)
- Storm lift and filter operations using regular expressions (~=) are now case insensitive by default. (#3403)
- Add a `unique()` method to the Storm `list` object. This returns a new list with only unique elements in it. (#3415)
- Add support for `synapse.tools.autodoc` to generate documentation for API definitions declared in Storm packages. (#3382)
- A review of Storm library functions was performed and all `readonly` safe functions have been marked for execution in a `readonly` Storm runtime. (#3402)
- Allow setting the layers on a root View with forks. (#3413)

17.15.4 Bugfixes

- Per-node Storm variables are now passed into subquery assignment expressions. (#3405)
- Fix an issue with Storm Dmon hive storage being opened too late in the Cortex startup sequence. (#3411)
- Remove a check when deleting tags from a node which prevented tag deletion from a node when the root tag was deleted in a parent view. (#3408)

17.16 v2.153.0 - 2023-10-27

17.16.1 Model Changes

- Update to the `inet` and `ou` models.

  (#3393) (#3396)

  Deprecated Properties

  inet:web:acct
  The `inet:web:acct` form had the following properties marked as deprecated:

  - `name:en`
  - `realname:en`

  inet:web:group
  The `inet:web:group` form had the following property marked as deprecated:

  - `name:en`

  ou:industry
  The `ou:industry` form had the following property marked as deprecated:

  - `subs`

17.16.2 Features and Enhancements

- Add a new Storm API, `$lib.cortex.httpapi`, for creating and managing Extended HTTP API endpoints. These Cortex HTTP API endpoints allow a user to create custom responses via Storm. Documentation for this feature can be found at *Extended HTTP API*. (#3366)
- Add a new Storm API, `$lib.iters.zip()`, to iterate over sequences of items together. (#3392) (#3398)
- Add a Storm command `stats.countby` to tally occurrences of values and display a barchart representing the values. (#3385)
- Update the Storm command `auth.user.mod` to allow setting a user as admin on a specific auth gate. (#3391)
- The `proxy` argument to `$lib.inet.http.*`, `$lib.axon.wget()`, `$lib.axon.urlfile()`, and `$lib.axon.wput()` APIs is now gated behind the permission `storm.lib.inet.http.proxy`. Previously this required admin permission to utilize. (#3397)
- Add an `errors` parameter to `$lib.axon.readlines()`, `$lib.axon.csvrows()`, and `$lib.axon.jsonlines()`. This parameter defaults to `ignore` to ignore any decoding errors that are encountered when decoding text. (#3395)
- Lower the maximum allowed version of the `pyopenssl` library. (#3399)

17.16.3 Bugfixes

- Fix a bug in the `Cortex.syncLayersEvents()` and `Cortex.syncIndexEvents()` APIs which caused layers to stop sending their node edits under certain conditions. (#3394)
- Storm now raises a `BadSyntaxError` when attempting to filter by wildcard tags or tagprops when a value is specified for the filter. (#3373)

17.17 v2.152.0 - 2023-10-17

17.17.1 Model Changes

- Update to the `biz`, `crypto`, `geo`, `it`, `mat`, `media`, and `risk` models.

  (#3341) (#3377) (#3376) (#3381)

  Updated Interfaces

  `crypto:smart:effect`
  : Add a `doc` value to the interface.

  `it:host:activity`
  : Add a `doc` value to the interface.

  `taxonomy`
  : Add a `doc` value to the interface.

  Updated Types

  `time`
  : The `time` type now recognizes RFC822 formatted time strings.

  `biz:service:type:taxonomy`
  : The `taxonomy` interface has been added to the type.

geo:place:taxonomy
The taxonomy interface has been added to the type.

it:log:event:type:taxonomy
The taxonomy interface has been added to the type.

it:prod:soft:taxonomy
The taxonomy interface has been added to the type.

mat:type
The taxonomy interface has been added to the type.

media:news:taxonomy
The taxonomy interface has been added to the type.

risk:alert:taxonomy
The taxonomy interface has been added to the type.

risk:alert:verdict:taxonomy
The taxonomy interface has been added to the type.

risk:threat:type:taxonomy
The taxonomy interface has been added to the type.

New Forms

it:dev:repo:label
A developer selected label.

it:dev:repo:issue:label
A label applied to a repository issue.

17.17.2 Features and Enhancements

- Update the Storm string repr for `$lib.null` and `$lib.undef` values to `$lib.null` and `$lib.undef`. Previously these printed `None` and an opaque Python object repr. (#3361)
- The `synapse.tools.aha.list` CLI tool now checks if it is connected to an Aha server prior to enumerating Aha services. (#3371)

17.17.3 Bugfixes

- Update the `file:path` support for scrape related APIs to address an issue when matching against Linux style paths. (#3378)
- Update the `hex` type to `zeropad` strings prior to checking their validity. (#3387)
- Update the `yaml.CSafeLoader` check to not require the class to be available. (#3386)

17.17.4 Improved Documentation

- Update the documentation for the Storm `view.exec` command to explain the separation of events and nodes between the parent and sub-runtimes. (#3379)

17.18 v2.151.0 - 2023-10-06

17.18.1 Model Changes

- Update to the `it` model. (#3361)

  New Forms

  `it:mitre:attack:flow`
  A MITRE ATT&CK Flow diagram.

17.18.2 Features and Enhancements

- Add a new Storm library `$lib.infosec.mitre.attack.flow`. This can be used to normalize and create `it:mitre:attack:flow` nodes from MITRE ATT&CK Flow Diagrams. (#3361) (#3372)
- Update the Storm `note.add` command to set the `meta:note:created` property on the note. (#3569)
- Add the Axon HTTP APIs to the Cortex. These API endpoints use the Axon that the Cortex is configured to use. (#3550)
- Allow user defined functions in Storm to execute in a `readonly` Storm runtime. (#3552)
- Clarify the Nexus `IsReadOnly` exception to include the common cause for the error, which is normally insufficent space on disk. (#3359)
- Add a `SYN_LOG_DATEFORMAT` environment variable to allow specifying custom timestamp formats for Synapse services. (#3362)
- Add a `status` attribute to structured log events for user and role related log events. This attribute indicates if the event was a `CREATE`, `DELETE`, or `MODIFY` operation. (#3363)
- Update `Cell.getLogExtra()` to prefer using the `user` key from the task scope before using the `sess` key from the task scope. Cortex APIs which execute Storm queries now set the `user` scope to the user the query is running as. This increases the accuracy of log events caused by Storm queries when the `user` is specified in the `opts`. (#3356)
- Update Storm setitem AST operator to check the readonly flag on functions when operating in a `readonly` Storm runtime. (#3364)
- Update the minimum required version of the `fastjsonschema` library. (#3358)
- Update tests and remove the use of deprecated functions for improved Python 3.12 compatibility. (#3355) (#3567)

17.18.3 Bugfixes

- Fixed a bug when parenting a View to another View where the bottom view has more than one layer in it omitted non-write layers. The set of layers is now properly computed. (#3354)

17.18.4 Improved Documentation

- Update the list of Cortex permissions in the Admin Guide. (#3331)
- The Form documentation has been updated to project the secondary properties and associated light edges as tables. (#3348)

17.19 v2.150.0 - 2023-09-22

17.19.1 Model Changes

- Updates to the `inet` model. (#3347)

  Updated Types

  `inet:url`
  The `inet:url` type now recognizes UNC network paths and converts them into `smb://` URLs.

17.19.2 Features and Enhancements

- Allow Storm trigger APIs to reference triggers from other views. (#3342)
- Update the `synapse.lib.scrape` and associated APIs to capture additional data: (#3223) (#3347)

  `it:sec:cpe`
  CPE 2.3 strings are now identified.

  `inet:url`
  UNC based paths are now identified.

- Update the `synapse.lib.scrape` and associated APIs to use subprocesses when scraping large volumes of text. (#3344)
- Add additional logging for HTTP API endpoints when a request has invalid login information. (#3345)
- The CryoTank service has had permissions added to it. (#3328)

17.19.3 Bugfixes

- Stormtypes `stor` functions were not previously checked during `readonly` runtime execution. These are now validated and `stor` functions which would result in changing data in the Cortex will now raise an exception when used with a `readonly` Storm runtime. (#3349)

17.19.4 Improved Documentation

- Update the list of Cortex permissions in the Admin Guide. (#3331)
- The Form documentation has been updated to project the secondary properties and associated light edges as tables. (#3348)

17.20 v2.149.0 - 2023-09-14

17.20.1 Model Changes

- Updates to the `it`, `meta`, and `ou` models. (#3338)

  New Properties

  `taxonomy`
  : The interface had the following property added to it:

    `description`
    : A definition of the taxonomy entry.

  `inet:email:message`
  : The form had the following property added to it:

    `cc`
    : Email addresses parsed from the "cc" header.

  `meta:source`
  : The form had the following property added to it:

    `url`
    : A URL which documents the meta source.

  `ou:campaign`
  : The form had the following property added to it:

    `timeline`
    : A timeline of significant events related to the campaign.

  Deprecated Properties

  `taxonomy`
  : The `taxonomy` interface had the following property marked as deprecated:

    - `summary`

17.20.2 Features and Enhancements

- Add best-effort support to scrape APIs to identify Windows and Linux file paths. (#3343)
- Update the Storm `view.add` command to add a `--worldreadable` flag to create a view which is readable by the `all` role. The `$lib.view.add()` Storm API now also accepts an optional `worldreadable` argument as well. (#3333)
- Update the Storm `note.add` command to add a `--yield` flag which yields the newly created note. (#3337)
- Add Storm commands `gen.ou.id.number` and `gen.ou.id.type` to help generate `ou:id:number` and `ou:id:type` nodes. (#3339)
- Support dynamically setting a Layer to `readonly` using the Storm `$layer.set()` API. (#3332)

- Update the Storm command `help` to display information about Storm types, Storm Libraries and functions. (#3335)

17.20.3 Bugfixes

- Ensure that the Cell `tmp` directory is on the same volume as the Cell storage directory prior to attempting to run the onboot optimization process. If the volumes are different this now issues a warning message and skips the optimization process. (#3336)
- Protect the Cortex Cron scheduling loop from errors that could happen when starting an agenda item. (#3340)

17.21 v2.148.0 - 2023-09-05

17.21.1 Features and Enhancements

- Add a `$lib.jsonstor.cachedel()` API to allow for the removal of data created by `$lib.jsonstor.cacheget()`. (#3322)

17.21.2 Bugfixes

- Ensure the base Cell `fini()`'s the Aha client that it creates. This fixes a unit test performance issue. (#3324)

17.21.3 Deprecations

- Mark the following Cryotank related API arguments and functions as deprecated. These APIs are related to server-side offset tracking for callers. Code which relies on these should be updated to do local offset tracking. These APIs and arguments will be removed in v2.150.0. (#3326)
  - `CryoApi.puts(seqn=...)` argument.
  - `CryoApi.rows(seqn=...)` argument.
  - `CryoApi.slice(iden=...)` argument.
  - `CryoApi.offset()` function.
  - `CryoTank.getOffset()` function.
  - `CryoTank.setOffset()` function.
  - `CryoTank.puts(seqn=...)` argument.
  - `CryoTank.rows(seqn=...)` argument.
  - `CryoTank.slice(iden=...)` argument.
  - `TankAPI.offset()` function.
  - `TankApi.puts(seqn=...)` argument.
  - `TankAPI.slice(iden=...)` argument.

17.22 v2.147.0 - 2023-08-31

17.22.1 Features and Enhancements

- Add `wait` and `timeout` arguments to Cryotank `slice()` APIs. (#3320)
- Add a `charset` parameter to the Storm `inet:imap:server.search()` API. This can be used to specify the `CHARSET` value when crafting a search query. (#3318)

17.22.2 Bugfixes

- Vendor the `asyncio.timeouts.Timeout` class from Python 3.11.3 to ensure correct task cancellation behavior is available for `synapse.common.wait_for()`. (#3321)

17.23 v2.146.0 - 2023-08-29

17.23.1 Features and Enhancements

- Update Storm `graph` projection to only include edges between nodes in the result set and include a *"reverse": true* in the edge info when embedding an edge on its target node once it is yielded. (#3305)
- Map the Nexus LMDB slab with `map_async=True` by default. (#3314)
- Mark the Storm `macro.exec` as a `readonly` safe command. Mark the Storm APIs `$lib.macro.list()` and `$lib.macro.get()` as `readonly` safe. Mark the `str` APIs as `readonly` safe. (#3316)

17.23.2 Bugfixes

- Fix an issue where Layer data migrations failed when a layer was marked as `readonly`. (#3313)
- Fix an issue where utility functions for packed nodes in `synapse.lib.node` did not handle nodes from HTTP API endpoints. (#3315)

17.24 v2.145.0 - 2023-08-25

17.24.1 Automatic Migrations

- Update indexing for light edges to index the N1 and N2 node identifiers together. (#3302)
- See *Data Migration* for more information about automatic migrations.

17.24.2 Model Changes

- Update to the `inet`, `it`, and `meta` models. (#3285) (#3298) (#3301) (#3310)

  New Types

  `it:sec:tlp`
  : The US CISA Traffic-Light-Protocol used to designate information sharing boundaries.

  `meta:priority`
  : A generic priority enumeration.

  `meta:severity`
  : A generic severity enumeration.

  New Forms

  `it:sec:metrics`
  : A node used to track metrics of an organization's infosec program.

  `it:sec:vuln:scan`
  : An instance of running a vulnerability scan.

  `it:sec:vuln:scan:result`
  : A vulnerability scan result for an asset.``

  New Properties

  `it:dev:repo:issue`
  : The form had the following properties added to it:

    `updated`
    : The time the issue was updated.

    `id`
    : The ID of the issue in the repository system.

  `it:dev:repo:issue:comment`
  : The form had the following properties added to it:

    `created`
    : The time the comment was created.

    `updated`
    : The time the comment was updated.

  `it:dev:repo:diff:comment`
  : The form had the following properties added to it:

    `created`
    : The time the comment was created.

    `updated`
    : The time the comment was updated.

  `meta:note`
  : The form had the following properties added to it:

    `updated`
    : The time the note was updated.

  Deprecated Properties

  `it:exec:proc`
  : The `it:exec:proc` form had the following property marked as deprecated:

- src:exe

inet:whois:iprec
The inet:whois:iprec form had the following property marked as deprecated:

- registrant

17.24.3 Features and Enhancements

- Add a new Storm keyword, reverse(...), which can be used to run a lift operation in reverse order. (#3266)
- Update indexing for light edges to index the N1 and N2 node identifiers together. (#3302)
- Update the Storm once command behavior and documentation to be more intuitive when setting its timestamp and allowing nodes through it. (#3282)
- Add a synapse_version key to the Storm Package schema. This can be used to provide a string version identifier with a minimum and maximum version, such as >=2.145.0,<3.0.0. (#3304)
- Update the Storm runtime to respect permissions declared with a default value of true. This allows Storm packages to define permissions which are defaulted to true. (#3287)
- Add a SIGHUP handler to the base Cell which can be used to reload HTTPS certificate files from disk. The synapse.tools.reload tool can also be used to trigger this behavior. (#3293)
- The optional max:users feature no longer counts locked or archived users when adding users. (#3295)
- Update the YAML functions to use the yaml.CSafeLoader and yaml.CSafeDumper. (#3289)

17.24.4 Bugfixes

- Replace asyncio.wait_for() use with a copy of the Python 3.12 implementation to avoid a race condition when cancelling tasks. (#3299) (#3307)
- Fix an issue with the Storm trigger set() method not properly checking the values that it allows to be set. (#3290)
- Fix an off-by-one bug in the SlabSeqn.aiter() method. (#3300)
- Fix a performance issue with the IPv6 regular expression used in the scrape APIs. (#3311)

17.24.5 Improved Documentation

- Revise the Storm User Guide to consolidate the background information and data modeling sections. Add a user focused section on Views and Layers. (#3303)
- Add int type specific information to the Storm documentation. (#3288)
- The Storm movetag command now moves the doc:url property from the old syn:tag node to the new syn:tag node. (#3294)
- Storm Library and Type documentation no longer renders function signatures with Python style defaults. (#3296)

17.24.6 Deprecations

- Many deprecated Cortex and splice related APIs have been marked for removal after 2023-10-01. The full list of APIs which will be removed can be found at *API Deprecation Notice - 2023-10-01*. (#3292)
- The use of `synapse.common.aclosing()` has been replaced with `contextlib.aclosing()`. The vendored `aclosing()` implementation will be removed in `v2.250.0`. (#3206)

17.25 v2.144.0 - 2023-08-09

17.25.1 Model Changes

- Updates to the `inet:dns` and `it` model. (#3257) (#3276)

  New Forms

  `it:dev:repo:type:taxonomy`
  A version control system type taxonomy.

  `it:dev:repo`
  A version control system instance.

  `it:dev:repo:remote`
  A remote repo that is tracked for changes/branches/etc.

  `it:dev:repo:branch`
  A branch in a version control system instance.

  `it:dev:repo:commit`
  A commit to a repository.

  `it:dev:repo:diff`
  A diff of a file being applied in a single commit.

  `it:dev:repo:issue`
  An issue raised in a repository.

  `it:dev:repo:issue:comment`
  A comment on an issue in a repository.

  `it:dev:repo:diff:comment`
  A comment on a diff in a repository.

  New Properties

  `inet:dns:answer`
  The form had the following properties added to it:

  `time`
  The time that the DNS response was transmitted.

17.25.2 Features and Enhancements

- The iden of the Cron job is now added to the Storm query log made with the `synapse.storm` logger when using structured logging. (#3235)
- Add a `keepalive` option to the Storm query `opts`. This may be used with long-running Storm queries when behind a network proxy or load balancer which may terminate idle connections. (#3272)
- Update the allowed versions of the `cryptography` library. (#3281)

17.25.3 Bugfixes

- Fix an issue where Storm Dmons could start prior to data model migrations. (#3279)
- Adjust the storage convention for `once` state data to fix an edge case and clarify documentation. (#3282)
- Fix an issue with missing keys in storage nodes during migrations. (#3284)

17.26 v2.143.0 - 2023-07-28

17.26.1 Model Changes

- Update to the `crypto` model. (#3256)

  Updated Types

  `hex`
  : The `zeropad` option has been changed from a `bool` to an `int`. It may now be used to specify the zero extended length of the hex string.

  Updated Properties

  `crypto:x509:cert`
  : The form had the following properties updated on it:

    `serial`
    : The `size` value has been changed to `zeropad` to zeropad values with less than 40 octets, and to allow storing large serial numbers from malformed certificates.

17.26.2 Features and Enhancements

- Add `$lib.model.ext.getExtModel()` and `$lib.model.ext.addExtModel()` Storm APIs to get all the extended model definitions in a Cortex and to add extended model definitions to a Cortex in bulk. (#3252)
- Add `inet:ipv6` to the list of types identified with scrape APIs. The `inet:server` form identified by scrape APIs now also identifies IPv6 server addresses. (#3259)
- Add a check to the Cortex startup to identify and log the presence of deprecated model elements and direct users to check and lock them at *model.deprecated.check*. (#3253) (#3264)
- Add a new Storm function, `$lib.vars.type()`, to get the type value of an object. (#3100)
- Add a Storm library, `$lib.pack`, for packing and unpacking structured byte values. (#3261)
- The Storm `$lib.gen()` functions and associated commands now generate stable guid values based on their inputs when making nodes. (#3268)
- Add the `.bazar` TLD to the list of TLDs identified by the Synapse scrape functionality. (#3271)

- Add the View iden to the task identifier for running Storm tasks. (#3247)
- Add performance related sysctl values to the output of the Storm `Cell.getSystemInfo()` and `$lib.cell.getSystemInfo()` APIs. (#3236)
- Update the allowed versions of the `vcrpy` library. Thank you `captainGeech42` for the contribution. (#3204)

17.26.3 Bugfixes

- Ensure the input to the `CoreAPI.storm()` (and related APIs) is a string. (#3255) (#3269)
- Fix a bug in `synapse.tools.aha.enroll` where a user with a `telepath.yaml` file containing an `aha:servers` key with a list of lists failed to enroll a local user. (#3260)
- Fix an issue where Storm functions using `emit` failed to cleanup their sub-runtimes. (#3250)
- Add verification that a Storm function call is being made on a callable object and raise a `StormRuntimeError` if the object cannot be called. Previously invalid calls could raise a `TypeError`. (#3243)
- Fix the order of the Beholder `cron:stop` message firing when a Cron job is stopped. (#3265)

17.26.4 Improved Documentation

- Add a section to the Storm reference for user defined functions in Storm. That can be found at *Storm Reference - Advanced - Functions*. (#3245)
- Update the devops documentation to add a note about the Telepath `aha://` protocol using a `mirror=true` parameter to connect to a service mirror instead of a leader. (#3267)
- Update the `preboot.sh` example script to account for Docker changes introduced in `v2.133.0`.

17.27 v2.142.2 - 2023-07-19

17.27.1 Bugfixes

- Fix an issue which caused the Docker image tags for `vertexproject/synapse-cryotank:v2.141.1`, `vertexproject/synapse-jsonstor:v2.141.1`, and `vertexproject/synapse-stemcell:v2.141.1`, to refer to same image. (#3249)

17.28 v2.142.1 - 2023-07-19

17.28.1 Bugfixes

- Fix an issue which prevented the publication of the Synapse containers with `v2.x.x` tags. (#3248)

17.29 v2.142.0 - 2023-07-19

17.29.1 Automatic Migrations

- Renormalize the `risk:vuln:cvss:v2` and `risk:vuln:cvss:v3` properties. (#3224)
- Migrate the `risk:vuln:name` type from a `str` to a `risk:vulnname` form. (#3227)
- See *Data Migration* for more information about automatic migrations.

17.29.2 Model Changes

- Update to the `it`, `ou`, and `risk` models. (#3224) (#3227) (#3237)

  New Forms

  `risk:vulnname`
  Add a form to capture vulnerability name such as log4j or rowhammer.

  Updated Types

  `hex`
  The `hex` base type now accepts a `zeropad` option that can be used to zero-extend a hex string during normalization.

  `cvss:v2`
  The type now accepts and normalizes unordered CVSS vectors.

  `cvss:v3`
  The type now accepts and normalizes unordered CVSS vectors.

  New Properties

  `it:sec:c2:config`
  The form had the following properties added to it:

  `decoys`
  An array of URLs used as decoy connections to obfuscate the C2 servers.

  `ou:technique`
  The form had the following properties added to it:

  `reporter`
  The organization reporting on the technique.

  `reporter:name`
  The name of the organization reporting on the technique.

  `risk:vuln`
  The form had the following properties added to it:

  `names`
  An array of alternate names for the vulnerability.

17.29.3 Features and Enhancements

- Always convert dictionary keys to their primitive values when working with dictionary objects in Storm. Dictionary objects can no longer have keys set which are mutable objects, such as Nodes. (#3233)
- Add support for octal constants, such as `0o755`, in Storm expressions. (#3231)
- Add additional events to the Behold API message stream for the addition and removal of extended model elements. (#3228)
- Update the `$lib.dmon.add()` variable capture to record variables from embedded query objects. (#3230)
- Add a `.title()` method on Storm strings to get title case formatted strings. (#3242)
- Add a general purpose process pool using forked workers in order to speed up certain processing operations. This includes the Storm operations for JSONSchema parsing, HTML parsing, STIX validation, and XML parsing. (#3033) (#3229)
- Add a new Cell configuration option, `max:users`. This can be set to limit the maximum number of non-`root` users on Cell. (#3244)
- Add an `/api/v1/aha/services` HTTP API to the Aha service. This can be used to get a list of the services registered with Aha. (#3238)
- Add support for Cosign signatures of tagged Synapse containers. See additional information at *Verifying container image signatures*. (#3196)
- Adjust internal names for Storm objects. (#3229)

17.29.4 Bugfixes

- Fix a bug in the scrape for `inet:ipv4` where IP addresses were found when there was leading or trailing numbers around the IP addresses. (#3234)
- Fix a bug where `$lib.model.ext.delForm()` did not check for extended property definitions before deletion. Extended properties on a custom form must be deleted prior to deleting the form. (#3223)
- Always remove the `mirror` configuration option from `cell.yaml` file when provisioning a service via Aha. The previous behavior prevented the correct restoration of a service from a backup which was previously provisioned as a mirror and is being restored as a leader. (#3240)
- Add additional type checking when adding extended model forms and properties to the Cortex. Previously invalid types could raise an `AttributeError`. (#3243)

17.29.5 Improved Documentation

- Update the Storm lift reference to add an example of lifting nodes by the universal `.created` property. (#3245)

17.30 v2.141.0 - 2023-07-07

17.30.1 Model Changes

- Update to the `it` and `lang` models. (#3219)

  New Properties

  `it:host`
    The form had the following properties added to it:

    `keyboard:language`
      The primary keyboard input language configured on the host.

    `keyboard:layout`
      The primary keyboard layout configured on the host.

  `lang:language`
    The form had the following property added to it:

    `code`
      The language code for this language.

17.30.2 Features and Enhancements

- Update `$lib.infosec.cvss.vectToScore()` to include a normalized CVSS vector in the output. (#3211)
- Optimize the addition and removal of lightweight edges when operating on N1 edges in Storm. (#3214)
- Added `$lib.gen.langByCode`. (#3219)

17.30.3 Bugfixes

- Fix bug with regular expression comparisons for some types. (#3213)
- Fix a `TypeError` being raised when passing a heavy Number object to `$lib.math.number()`. (#3215)
- Fix an issue with the Cell backup space checks. They now properly calculate the amount of free space when the Cell backup directory is configured on a separate volume from the Cell storage directory. (#3216)
- Prevent the `yield` operator from directly emitting nodes into the Storm pipeline if those node objects came from a different view. Nodes previously lifted in this manner must be lifted by calling the `iden()` function on the object to ensure the node being lifted into the pipeline reflects the current view. (#3218)
- Always remove the `mirror` configuration option from `cell.mods.yaml` when provisioning a service via Aha. The previous behavior prevented the correct restoration of a service from a backup which had been changed from being a leader to being a mirror. (#3220)

17.31 v2.140.1 - 2023-06-30

17.31.1 Bugfixes

- Fix a typo which prevented the Synapse package for `v2.140.0` from being published on PyPI. (#3212)

17.32 v2.140.0 - 2023-06-30

17.32.1 Announcement

Synapse now only supports Python 3.11+.

17.32.2 Model Changes

- Update to the `inet`, `file`, and `ou` models. (#3192) (#3202) (#3207)

  New Types

  `file:archive:entry`
  : Add a type to capture an archive entry representing a file and metadata from within a parent archive file.

  Updated Types

  `time`
  : Time values with precision beyond milliseconds are now truncated to millsecond values.

  `hex`
  : Hex types now have whitespace and colon (`:`) characters stripped from them when lifting and normalizing them.

  `inet:ipv6`
  : Add comparators for >=, >, <=, < operations when lifting and filtering IPV6 values.

  `ou:naics`
  : Update the type to allow recording NIACS sector and subsector prefixes.

17.32.3 Features and Enhancements

- Synapse now only supports Python 3.11+. The library will now fail to import on earlier Python interpeters, and the published modules on PyPI will no longer install on Python versions < 3.11. (#3156)
- Replace `setup.py` with a `pyproject.toml` file. (#3156) (#3195)
- Usages of `hashlib.md5()` and `hashlib.sha1()` have been updated to add the `usedforsecurity=False` argument. (#3163)
- The Storm `diff` command is now marked as safe for `readonly` execution. (#3207)
- Add a `svc:set` event to the Behold API message stream. This event is fired when a Cortex connects to a Storm Service. (#3205)

17.32.4 Bugfixes

- Catch `ZeroDivisionError` and `decimal.InvalidOperation` errors in Storm expressions and raise a `StormRuntimeError`. (#3203)
- Fix a bug where `synapse.lib.platforms.linux.getTotalMemory()` did not return the correct value in a process running in cgroupsv1 without a maximum memory limit set. (#3198)
- Fix a bug where a Cron job could be created with an invalid Storm query. Cron jobs now have their queries parsed as part of creation to ensure that they are valid Storm. `$lib.cron` APIs now accept heavy Storm query objects as query inputs. (#3201) (#3207)
- Field data sent via Storm `$lib.inet.http` APIs that uses a multipart upload without a valid `name` field now raises a `BadArg` error. Previously this would result in a Python `TypeError`. (#3199) (#3206)

17.32.5 Deprecations

- Remove the deprecated `synapse.common.lockfile()` function. (#3191)

17.33 v2.139.0 - 2023-06-16

17.33.1 Announcement

Due to the introduction of several powerful new APIs and performance improvements, Synapse will be updating to *only* support Python >=3.11. Our current plan is to drop support for Python <=3.10 in ~4 weeks on 2023-06-19. The next release after 2023-06-19 will include changes that are not backward compatible to earlier versions of Python.

If you currently deploy Synapse Open-Source or Synapse Enterprise via the standard docker containers, you will be unaffected. If you install Synapse via PyPI, you will need to ensure that your environment is updated to Python 3.11+.

17.33.2 Model Changes

- Update `it:sec:cpe` normalization to extend truncated CPE2.3 strings. (#3186)

17.33.3 Features and Enhancements

- The `str` type now accepts `float` values to normalize. (#3174)

17.33.4 Bugfixes

- Fix an issue where the `file:bytes:sha256` property set handler could fail during data merging. (#3180)
- Fix an issue where iterating light edges on nodes could result in degraded Cortex performance. (#3186)

17.33.5 Improved Documentation

- Update the Cortex admin guide to include additional examples for setting up user and role permissions. (#3187)

17.34 v2.138.0 - 2023-06-13

17.34.1 Features and Enhancements

- Add `it:sec:cwe` to the list of types identified with scrape APIs. (#3182)
- Update the calculations done by `$lib.infosec.cvss.vectToScore()` to more closely emulate the NVD CVSS calculator. (#3181)

17.34.2 Bugfixes

- Fix an issue with `synapse.tools.storm` where the `!export` command did not use the view specified when starting the tool. (#3184)
- The `synapse.common.getSslCtx()` API now only attempts to load files in the target directory. This avoids confusing errors that may be logged when the target directory contains sub directories. (#3179)
- Fix an edge case in `$lib.infosec.cvss.vectToScore()` when calculating CVSS v2 scores. (#3181)

17.34.3 Deprecations

- Mark the Python function `synapse.common.lockfile()` as deprecated. It will be removed in `v2.140.0`. (#3183)

17.35 v2.137.0 - 2023-06-09

17.35.1 Automatic Migrations

- Migrate any `inet:url` nodes with `:user` and `:passwd` properties which may have been URL encoded. These values are now decoded. (#3169)
- Migrate the storage type for the `file:bytes:mime:pe:imphash` property. (#3173)
- See *Data Migration* for more information about automatic migrations.

17.35.2 Model Changes

- Updates to the `geospace`, `inet`, `infotech`, `ou`, `risk`, and `transport` models. (#3169)

  New Types

  `it:mitre:attack:matrix`
  Add a type to capture the enumeration of MITRE ATT&CK matrix values.

  New Forms

  `inet:egress`
  Add a form to capture a host using a specific network egress client address.

`it:prod:softreg`
Add a form to capture a registry entry is created by a specific software version.

`transport:land:vehicle`
Add a form to capture an individual vehicle.

`transport:land:registration`
Add a form to capture the registration issued to a contact for a land vehicle.

`transport:land:license`
Add a form to capture the license to operate a land vehicle issued to a contact.

New Properties

`inet:http:request`
The form had the following property added to it:

`referer`
The referer URL parsed from the "Referer:" header in the request.

`inet:search:query`
The form had the following property added to it:

`request`
The HTTP request used to issue the query.

`it:mitre:attack:tactic`
The form had the following property added to it:

`matrix`
The ATT&CK matrix which defines the tactic.

`it:mitre:attack:technique`
The form had the following property added to it:

`matrix`
The ATT&CK matrix which defines the technique.

`it:mitre:attack:mitigation`
The form had the following property added to it:

`matrix`
The ATT&CK matrix which defines the mitigation.

`it:app:snort:rule`
The form had the following property added to it:

`engine`
The snort engine ID which can parse and evaluate the rule text.

`it:app:yara:rule`
The form had the following properties added to it:

`ext:id`
The YARA rule ID from an external system.

`url`
A URL which documents the YARA rule.

`ou:campaign`
The form had the following property added to it:

`tag`
The tag used to annotate nodes that are associated with the campaign.

`ou:org`
The form had the following properties added to it:

`country`
The organization's country of origin.

`country:code`
The 2 digit ISO 3166 country code for the organization's country of origin.

`risk:threat`
The form had the following properties added to it:

`country`
The reporting organization's assessed country of origin of the threat cluster.

`country:code`
The 2 digit ISO 3166 country code for the threat cluster's assessed country of origin.

`risk:compromise`
The form had the following property added to it:

`vector`
The attack assessed to be the initial compromise vector.

Light Edges

`detects`
When used with a `meta:rule` node, the edge indicates the rule was designed to detect instances of the target node.

When used with an `it:app:snort:rule` node, the edge indicates the rule was designed to detect instances of the target node.

When used with an `it:app:yara:rule` node, the edge indicates the rule was designed to detect instances of the target node.

`contains`
When used between two `geo:place` nodes, the edge indicates the source place completely contains the target place.

Deprecated Properties

`geo:place`
The form had the following property marked as deprecated:

- `parent`

17.35.3 Features and Enhancements

- Add a modulo arithmetic operator (`%`) to Storm expression parsing. (#3168)
- Add `$lib.auth.easyperm` Storm library for interacting with objects that use a simplified permissions model. (#3167)
- Add `.vars` attribute to the Storm `auth:user` object. This can be used to access user variables. (#3167)
- Add `$lib.infosec.cvss.vectToScore()` to calculate CVSS scores. (#3171)
- The Storm `delnode` command node now requires the use of `--force` to delete a node which has lightweight edges pointing to it. (#3176)
- The STIX export configuration may now include a `synapse_extension` value set to `$lib.false` to disable the Synapse STIX extension data from being added to objects in the bundle. (#3177)

- Remove whitespace stripping from Storm queries prior to parsing them. This allows any error highlighting information to accurately reflect the query submitted to the Cortex. (#3175)

17.35.4 Bugfixes

- Fix an issue where raising an integer value to a fractional power in Storm was not handled correctly. (#3170)
- Handle a SyntaxError that may occur during Storm parsing due to a change in CPython 3.11.4. (#3170)
- The `inet:url` type now URL decodes the `user` and `passwd` properties when normalizing them. Thank you `captainGeech42` for the bug report. (#2568) (#3169)
- The URL parser in `synapse.lib.urlhelp` now URL decodes the `user` and `passwd` values when parsing URLs. (#3178)

17.35.5 Deprecations

- Mark the Storm functions `$lib.infosec.cvss.saveVectToNode()` and `$lib.infosec.cvss.vectToProps()` as deprecated. (#3178)

17.36 v2.136.0 - 2023-06-02

17.36.1 Model Changes

- Boolean values in the Synapse model now have lowercase `true` and `false` repr values. (#3159)
- The trailing `.` on the taxonomy repr has been removed. (#3159)

17.36.2 Features and Enhancements

- Normalize tag names when performing lift and filter operations. (#3094)
- Add `$lib.compression.bzip2`, `$lib.compression.gzip`, and `$lib.compression.zlib` Storm libraries to assist with compressing and decompressing bytes. (#3155) (#3162)
- Add a new Cell configuration option, `https:parse:proxy:remoteip`. When this is set to `true`, the Cell HTTPS server will parse `X-Forwarded-For` and `X-Real-IP` headers to determine the remote IP of an request. (#3160)
- Update the allowed versions of the `fastjsonschema` and `pycryptodome` libraries. Update the required version of the `vcrpy` library to account for changes in `urllib3`. Remove the pinned requirement for the `requests` library. (#3164)

17.36.3 Bugfixes

- Prevent zero length tag lift operations. (#3094)
- Fix an issue where tag properties with the type `ival`, or `time` types with `ismin` or `ismax` options set, were not properly merged when being set. (#3161)
- Fix a missing `mesg` value on `NoSuchForm` exception raised by the `layer liftByTag()` API. (#3165)

17.37 v2.135.0 - 2023-05-24

17.37.1 Features and Enhancements

- Add a `--index` option to the Storm `auth.user.grant` command. (#3150)
- Add additional type handling in the Storm view and layer `set()` APIs. (#3147)
- Add a new Storm command, `auth.perms.list`, to list all of the permissions registered with the Cortex. (#3135) (#3154)

17.37.2 Bugfixes

- Fix an issue where attempting a tag lift with a variable containing a zero-length string would raise an MDB error. (#3094)
- Fix an issue in the Axon `csvrows()` and `readlines()` APIs where certain exceptions would not be raised. (#3141)
- Fix an issue with the Storm `runas` command which prevented it being used with a privileged Storm runtime. (#3147)
- Fix support for Storm list objects in `$lib.max()` and `$lib.min()`. (#3153)

17.37.3 Improved Documentation

- Update the Cortex admin guide to include the output of the `auth.perms.list` command. (#3135)

17.38 v2.134.0 - 2023-05-17

17.38.1 Model Changes

- Updates to the `risk` model. (#3137)

  Light Edges

  `addresses`

  When used with a `risk:mitigation` and a `ou:technique` node, the edge indicates the mitigation addresses the technique.

17.38.2 Features and Enhancements

- Add a `--forms` option to the Storm `scrape` command. This can be used to limit the forms that are made from scraping the input text. The `scrape` command now uses the View scrape interface to generate its matches, which may include scrape functionality added via power-ups. The `scrape` command no longer produces warning messages when matched text is not valid for making nodes. (#3127)
- Add a `revs` definition to the STIX export configuration, to allow for adding in reverse relationships. (#3137)
- Add a `--delbytes` option to the Storm `delnode` command. This can be used to delete the bytes from an Axon when deleting a `file:bytes` node. (#3140)
- Add support for printing nice versions of the Storm `model:form`, `model:property`, `model:tagprop`, and `model:type` objects. (#3134) (#3139)

17.38.3 Bugfixes

- Fix an exception that was raised when setting the parent of a View. (#3131) (#3132)
- Fix an issue with the text scrape regular expressions misidentifying the `ftp://` scheme. (#3127)
- Correctly handle `readonly` properties in the Storm `copyto` command. (#3142)
- Fix an issue were partial service backups were not able to be removed. (#3143) (#3145)

17.39 v2.133.1 - 2023-05-09

17.39.1 Bugfixes

- Fix an issue where the Storm query hashing added in `v2.133.0` did not account for handling erroneous surrogate pairs in query text. (#3130)

17.39.2 Improved Documentation

- Update the Storm API Guide to include the `hash` key in the `init` message. (#3130)

17.40 v2.133.0 - 2023-05-08

17.40.1 Model Changes

- Updates to the `risk` model. (#3123)

  New Properties

  `risk:vuln`

  The `risk:vuln` form had the following properties added to it:

  `cvss:v2`

  The CVSS v2 vector for the vulnerability.

  `cvss:v2_0:score`

  The CVSS v2.0 overall score for the vulnerability.

`cvss:v2_0:score:base`
The CVSS v2.0 base score for the vulnerability.

`cvss:v2_0:score:temporal`
The CVSS v2.0 temporal score for the vulnerability.

`cvss:v2_0:score:environmental`
The CVSS v2.0 environmental score for the vulnerability.

`cvss:v3`
The CVSS v3 vector for the vulnerability.

`cvss:v3_0:score`
The CVSS v3.0 overall score for the vulnerability.

`cvss:v3_0:score:base`
The CVSS v3.0 base score for the vulnerability.

`cvss:v3_0:scare:temporal`
The CVSS v3.0 temporal score for the vulnerability.

`cvss:v3_0:score:environmental`
The CVSS v3.0 environmental score for the vulnerability.

`cvss:v3_1:score`
The CVSS v3.1 overall score for the vulnerability.

`cvss:v3_1:score:base`
The CVSS v3.1 base score for the vulnerability.

`cvss:v3_1:scare:temporal`
The CVSS v3.1 temporal score for the vulnerability.

`cvss:v3_1:score:environmental`
The CVSS v3.1 environmental score for the vulnerability.

Deprecated Properties

`risk:vuln`
The `risk:vuln` form had the following properties marked as deprecated:

- `cvss:av`
- `cvss:ac`
- `cvss:pr`
- `cvss:ui`
- `cvss:s`
- `cvss:c`
- `cvss:i`
- `cvss:a`
- `cvss:e`
- `cvss:rl`
- `cvss:rc`
- `cvss:mav`
- `cvss:mac`

- `cvss:mpr`
- `cvss:mui`
- `cvss:ms`
- `cvss:mc`
- `cvss:mi`
- `cvss:ma`
- `cvss:cr`
- `cvss:ir`
- `cvss:ar`
- `cvss:score`
- `cvss:score:temporal`
- `cvss:score:environmental`

17.40.2 Features and Enhancements

- Update the base Synapse images to use Debian bookworm and use Python 3.11 as the Python runtime. For users which build custom images from our published images, see additional information at *Working with Synapse Images* for changes which may affect you. (#3025)
- Add a `highlight` parameter to BadSyntaxError and some exceptions raised during the execution of a Storm block. This contains detailed information about where an error occurred in the Storm code. (#3063)
- Allow callers to specify an `iden` value when creating a Storm Dmon or a trigger. (#3121)
- Add support for STIX export configs to specify pivots to include additional nodes. (#3122)
- The Storm `auth.user.addrule` and `auth.role.addrule` now have an optional `--index` argument that allows specifying the rule location as a 0-based index value. (#3124)
- The Storm `auth.user.show` command now shows the user's `admin` status on authgates. (#3124)
- Add a `--only-url` flag to the `synapse.tools.aha.provision.service` and `synapse.tools.aha.provision.user` CLI tools. When set, the tool only prints the URL to stdout. (#3125)
- Add additional layer validation in the View schema. (#3128)
- Update the allowed version of the `cryptography`, `coverage`, `idna`, `pycryptodome`, `python-bitcoin`, and `vcrpy` libraries. (#3025)

17.40.3 Bugfixes

- Ensure the CLI tools `synapse.tools.cellauth`, `synapse.tools.csvtool`, and `synapse.tools.easycert` now return 1 on an execution failure. In some cases they previously returned `-1`. (#3118)

17.41 v2.132.0 - 2023-05-02

17.41.1 Features and Enhancements

- Update the minimum required version of the `fastjsonschema`, `lark`, and `pytz` libraries. Update the allowed version of the `packaging` and `scalecodec` libraries. (#3118)

17.41.2 Bugfixes

- Cap the maximum version of the `requests` library until downstream use of that library has been updated to account for changes in `urllib3`. (#3119)
- Properly add parent scope vars to `background` command context. (#3120)

17.42 v2.131.0 - 2023-05-02

17.42.1 Automatic Migrations

- Migrate the `ou:campaign:name` property from a `str` to an `ou:campname` type and create the `ou:campname` nodes as needed. (#3082)
- Migrate the `risk:vuln:type` property from a `str` to a `risk:vuln:type:taxonomy` type and create the `risk:vuln:type:taxonomy` nodes as needed. (#3082)
- See *Data Migration* for more information about automatic migrations.

17.42.2 Features and Enhancements

- Updates to the `dns`, `inet`, `it`, `ou`, `ps`, and `risk` models. (#3082) (#3108) (#3113)

  `inet:dns:answer`
  Add a `mx:priority` property to record the priority of the MX response.

  `inet:dns:dynreg`
  Add a form to record the registration of a domain with a dynamic DNS provider.

  `inet:proto`
  Add a form to record a network protocol name.

  `inet:web:attachment`
  Add a form to record the instance of a file being sent to a web service by an account.

  `inet:web:file`
  Deprecate the `client`, `client:ipv4`, and `client:ipv6` properties in favor of using `inet:web:attachment`.

  `inet:web:logon`
  Remove incorrect `readonly` markings for properties.

  `it:app:snort:rule`
  Add an `id` property to record the snort rule id. Add an `author` property to record contact information for the rule author. Add `created` and `updated` properties to track when the rule was created and last updated. Add an `enabled` property to record if the rule should be used for snort evaluation engines. Add a `family` property to record the software family the rule is designed to detect.

`it:prod:softid`
: Add a form to record an identifier issued to a given host by a specific software application.

`ou:campname`
: Add a form to record the name of campaigns.

`ou:campaign`
: Change the `name` and `names` secondary properties from `str` to `ou:campname` types.

`ps:contact`
: Add a `place:name` to record the name of the place associated with the contact.

`risk:threat`
: Add an `active` property to record the interval of time when the threat cluster is assessed to have been active. Add a `reporter:published` property to record the time that a reporting organization first publicly disclosed the threat cluster.

`risk:tool:software`
: Add a `used` property to record the interval when the tool is assessed to have been deployed. Add a `reporter:discovered` property to record the time that a reporting organization first discovered the tool. Add a `reporter:published` property to record the time that a reporting organization first publicly disclosed the tool.

`risk:vuln:soft:range`
: Add a form to record a contiguous range of software versions which contain a vulnerability.

`risk:vuln`
: Change the `type` property from a `str` to a `risk:vuln:type:taxonomy`.

`risk:vuln:type:taxonomy`
: Add a form to record a taxonomy of vulnerability types.

- Add a new Storm command, `auth.user.allowed` that can be used to check if a user is allowed to use a given permission and why. (#3114)
- Add a new Storm command, `gen.ou.campaign`, to assist with generating or creating `ou:campaign` nodes. (#3082)
- Add a boolean `default` key to the permissions schema definition. This allows a Storm package permission to note what its default value is. (#3099)
- Data model migrations which fail to normalize existing secondary values into their new types now store those values in Node data on the affected nodes and remove those bad properties from the affected nodes. (#3117)

17.42.3 Bugfixes

- Fix an issue with the search functionality in our documentation missing the required jQuery library. (#3111)
- Unique nodes when performing multi-layer lifts on secondary properties without a value. (#3110)

17.42.4 Improved Documentation

- Add a section about managing data model deprecations to the Synapse Admin guide. (#3102)

17.42.5 Deprecations

- Remove the deprecated `synapse.lib.httpapi.HandlerBase.user()` and `synapse.lib.httpapi.HandlerBase.getUserBody()` functions. Remove the deprecated `synapse.axon.AxonFileHandler.axon()` function. (#3115)

17.43 v2.130.2 - 2023-04-26

17.43.1 Bugfixes

- Fix an issue where the `proxy` argument was not being passed to the Axon when attempting to post a file via Storm with the `$lib.inet.http.post()` API. (#3109)
- Fix an issue where adding a readonly layer that does not already exist would raise an error. (#3106)

17.44 v2.130.1 - 2023-04-25

17.44.1 Bugfixes

- Fix a race condition in a Telepath unit test which was happening during CI testing. (#3104)

17.45 v2.130.0 - 2023-04-25

17.45.1 Features and Enhancements

- Updates to the `infotech` model. (#3095)

  `it:host`

  Add an `ext:id` property for recording an external identifier for a host.
- Add support for deleting node properties by assigning `$lib.undef` to the property to be removed through `$node.props`. (#3098)
- The `Cell.ahaclient` is longer cached in the `synapse.telepath.aha_clients` dictionary. This isolates the Cell connection to Aha from other clients. (#3008)
- When the Cell mirror loop exits, it now reports the current `ready` status to the Aha service. This allows a service to mark itself as “not ready” when the loop restarts and it is a follower, since it may no longer be in the realtime change window. (#3008)
- Update the required versions of the `nbconvert`, `sphinx` and `hide-code` libraries used for building documentation. Increased the allowed ranges for the `pygments` and `jupyter-client` libraries. (#3103)

17.45.2 Bugfixes

- Fix an issue in backtick format strings where single quotes in certain positions would raise a syntax error. (#3096)
- Fix an issue where permissions were not correctly checked when assigning a property value through `$node.props`. (#3098)
- Fix an issue where the Cell would report a static `ready` value to the Aha service upon reconnecting, instead of the current `ready` status. The `Cell.ahainfo` value was replaced with a `Cell.getAhaInfo()` API which returns the current information to report to the Aha service. (#3008)

17.46 v2.129.0 - 2023-04-17

17.46.1 Features and Enhancements

- Updates to the `ou` and `risk` models. (#3080)

  `ou:campaign`
  Add a `names` property to record alternative names for the campaign. Add `reporter` and `reporter:name` properties to record information about a reporter of the campaign.

  `risk:attack`
  Add `reporter` and `reporter:name` properties to record information about a reporter of the attack.

  `risk:compromise`
  Add `reporter` and `reporter:name` properties to record information about a reporter of the compromise.

  `risk:vuln`
  Add `reporter` and `reporter:name` properties to record information about a reporter of the vulnerability.

- Add leader status to the `synapse.tools.aha.list` tool output. This will only be available if a leader has been registered for the service. (#3078)
- Add support for private values in Storm modules, which are specified by beginning the name with a double underscore (`__`). These values cannot be dereferenced outside of the module they are declared in. (#3079)
- Update error messages for Axon.wget, Axon.wput, and Axon.postfiles to include more helpful information. (#3077)
- Update `it:semver` string normalization to attempt parsing improperly formatted semver values. (#3080)
- Update Axon to always pass size value when saving bytes. (#3084)

17.46.2 Bugfixes

- Add missing `toprim()` calls on arguments to some `auth:user` and `auth:role` APIs. (#3086)
- Fix the regular expression used to validate custom STIX types. (#3093)

17.46.3 Improved Documentation

- Add sections on user and role permissions to the Synapse Admin guide. (#3073)

17.47 v2.128.0 - 2023-04-11

17.47.1 Automatic Migrations

- Migrate the `file:bytes:mime:pe:imphash` property from a `guid` to a `hash:md5` type and create the `hash:md5` nodes as needed. (#3056)
- Migrate the `ou:goal:name` property from a `str` to a `ou:goalname` type and create the `ou:goalname` nodes as needed. (#3056)
- Migrate the `ou:goal:type` property from a `str` to a `ou:goal:type:taxonomy` type and create the `ou:goal:type:taxonomy` nodes as needed. (#3056)
- See *Data Migration* for more information about automatic migrations.

17.47.2 Features and Enhancements

- Updates to the `belief`, `file`, `lang`, `it`, `meta`, `ou`, `pol`, and `risk` models. (#3056)

  `belief:tenet`
  : Add a `desc` property to record the description of the tenet.

  `file:bytes`
  : Change the type of the `mime:pe:imphash` from `guid` to `hash:md5`.

  `inet:flow`
  : Add a `raw` property which may be used to store additional protocol data about the flow.

  `it:app:snort:rule`
  : Add a `desc` property to record a brief description of the snort rule.

  `ou:goal`
  : Change the type of `name` from `str` to `ou:goalname`. Change the type of `type` from `str` to `ou:goal:type:taxonomy`. Add a `names` array to record alternative names for the goal. Deprecate the `prev` property in favor of types.

  `ou:goalname`
  : Add a form to record the name of a goal.

  `ou:goalname:type:taxonomy`
  : Add a taxonomy of goal types.

  `ou:industry`
  : Add a `type` property to record the industry taxonomy.

  `ou:industry:type:taxonomy`
  : Add a taxonomy to record industry types.

  `pol:immigration:status`
  : Add a form to track the immigration status of a contact.

  `pol:immigration:status:type:taxonomy`
  : Add a taxonomy of immigration types.

`risk:attack`
Add a `detected` property to record the first confirmed detection time of the attack. Add a `url` property to record a URL that documents the attack. Add a `ext:id` property to record an external identifier for the attack.

`risk:compromise`
Add a `detected` property to record the first confirmed detection time of the compromise.

- Add a Storm command `copyto` that can be used to create a copy of a node from the current view to a different view. (#3061)
- Add the current View iden to the structured log output of a Cortex executing a Storm query. (#3068)
- Update the allowed versions of the `lmdb`, `msgpack`, `tornado` and `xxhash` libraries. (#3070)
- Add Python 3.11 tests to the CircleCI configuration. Update some unit tests to account for Python 3.11 related changes. (#3070)
- Allow dereferencing from Storm expressions. (#3071)
- Add an `ispart` parameter to `$lib.tags.prefix` to skip `syn:tag:part` normalization of tag names. (#3074)
- Add `getEdges()`, `getEdgesByN1()`, and `getEdgesByN2()` APIs to the `layer` object. (#3076)

17.47.3 Bugfixes

- Fix an issue which prevented the `auth.user.revoke` Storm command from executing. (#3069)
- Fix an issue where `$node.data.list()` only returned the node data from the topmost layer containing node data. It now returns all the node data accessible for the node from the current view. (#3061)

17.47.4 Improved Documentation

- Update the Developer guide to note that the underlying Python runtime in Synapse images may change between releases. (#3070)

17.48 v2.127.0 - 2023-04-05

17.48.1 Features and Enhancements

- Set `Link` high water mark to one byte in preparation for Python 3.11 support. (#3064)
- Allow specifying dictionary keys in Storm with expressions and backtick format strings. (#3065)
- Allow using deref syntax (`*$form`) when lifting by form with tag (`*$form#tag`) and form with tagprop (`*$form#tag:tagprop`). (#3065)
- Add `cron:start` and `cron:stop` messages to the events emitted by the `behold()` API on the Cortex. These events are only emitted by the leader. (#3062)

17.48.2 Bugfixes

- Fix an issue where an Aha service running on a non-default port would not have that port included in the default Aha URLs. (#3049)
- Restore the `view.addNode()` Storm API behavior where making a node on a View object that corresponds to the currently executing view re-used the current Snap object. This allows nodeedits to be emitted from the Storm message stream. (#3066)

17.49 v2.126.0 - 2023-03-30

17.49.1 Features and Enhancements

- Add additional Storm commands to assist with managing Users and Roles in the Cortex. (#2923) (#3054)

  `auth.gate.show`
  Shows the definition for an AuthGate.

  `auth.role.delrule`
  Used to delete a rule from a Role.

  `auth.role.mod`
  Used to modify properties of a Role.

  `auth.role.del`
  Used to delete a Role.

  `auth.role.show`
  Shows the definition for a Role.

  `auth.role.list`
  List all Roles.

  `auth.user.delrule`
  Used to delete a rule from a User.

  `auth.user.grant`
  Used to grant a Role to a User.

  `auth.user.revoke`
  Used to revoke a Role from a User.

  `auth.role.mod`
  Used to modify properties of a User.

  `auth.user.show`
  Shows the definition of a User.

  `auth.user.list`
  List all Users.

- Update some of the auth related objects in Storm: (#2923)

  `auth:role`
  Add `popRule()` and `getRules()` functions. Add a `.gates` accessor to get all of the AuthGates associated with a role.

  `auth:user`
  Add `popRule()` and `getRules()` functions. Add a `.gates` accessor to get all of the AuthGates associated with a user.

- Add `$lib.auth.textFromRule()`, `$lib.auth.getPermDefs()` and `$lib.auth.getPermDef()` Storm library APIs to assist with working with permissions. (#2923)
- Add a new Storm library function, `$lib.iters.enum()`, to assist with enumerating an iterable object in Storm. (#2923)
- Update the `NoSuchName` exceptions which can be raised by Aha during service provisioning to clarify they are likely caused by re-using the one-time use URL. (#3047)
- Update `gen.ou.org.hq` command to set `ps:contact:org` if unset. (#3052)
- Add an `optional` flag for Storm package dependencies. (#3058)
- Add `.]`, `[.`, `http[:`, `https[:`, `hxxp[:` and `hxxps[:` to the list of known defanging strategies which are identified and replaced during text scraping. (#3057)

17.49.2 Bugfixes

- Fix an issue where passing a non-string value to `$lib.time.parse` with `errok=$lib.true` would still raise an exception. (#3046)
- Fix an issue where context managers could potentially not release resources after exiting. (#3055)
- Fix an issue where variables with non-string names could be passed into Storm runtimes. (#3059)
- Fix an issue with the Cardano regex used for scraping addresses. (#3057)
- Fix an issue where scraping a partial Cardano address could raise an error. (#3057)
- Fix an issue where the Storm API `view.addNode()` checked permissions against the incorrect authgate. This API now only returns a node if the View object is the same as the View the Storm query is executing in. (#3060)

17.49.3 Improved Documentation

- Fix link to Storm tool in Synapse Power-Ups section. (#3053)
- Add Kubernetes deployment examples, which show deploying Synapse services with Aha based provisioning. Add an example showing one mechanism to set `sysctl`'s in a managed Kubernetes deployment. (#3047)

17.50 v2.125.0 - 2023-03-14

17.50.1 Features and Enhancements

- Add a `size()` method on the STIX bundle object. (#3043)
- Update the minimum version of the `aio-socks` library to `0.8.0`. Update some unittests related to SOCKS proxy support to account for multiple versions of the `python-socks` library. (#3044)

17.50.2 Improved Documentation

- Update the Synapse documentation to add PDF and HTMLZip formats.

17.51 v2.124.0 - 2023-03-09

17.51.1 Features and Enhancements

- Added `--try` option to `gen.risk.vuln`, `gen.pol.country`, `gen.pol.country.government`, and `gen.ps.contact.email` commands and their associated Storm functions. (#3030)
- Added `$lib.gen.orgHqByName` and `$lib.gen.langByName`. (#3030)
- Added the configuration option `onboot:optimize` to all services to allow devops to delay service startup and allow LMDB to optimize storage for both size and performance. May also be set by environment variable `SYN_<SERVICE>_ONBOOT_OPTIMIZE=1` (#3001)
- Ensure that `AuthDeny` exceptions include the user iden in the `user` key, and the name in the `username` field. Previously the `AuthDeny` exceptions had multiple identifiers for these fields. (#3035)
- Add an optional `--view` argument to the `synapse.tools.storm` CLI tool. This allows a user to specify their working View for the Storm CLI. This was contributed by captainGeech42. (#2937)
- Updates to `synapse.lib.scope` and the `Scope` class. A `Scope.copy()` method has been added to create a shallow copy of a `Scope`. A module level `clone(task)` function has been added which clones the current task scope to the target `task`. Async Tasks created with `Base.schedCoro()` calls now get a shallow copy of the parent task scope. (#3021)
- Add a new Storm command, `batch`, to assist in processing nodes in batched sets. (#3034)
- Add global permissions, `` ` ``storm.macro.admin`` and `storm.macro.edit`, to allow users to administer or edit macros. (#3037)
- Mark the following Storm APIs as safe to execute in read-only queries: `$lib.auth.users.get()`, `$lib.auth.users.list()`, `$lib.auth.users.byname()`, `$lib.auth.roles.get()`, `$lib.auth.roles.list()`, `$lib.auth.roles.byname()`, `$lib.auth.gates.get()` and `$lib.auth.gates.list()`. (#3038)
- Added `uplink` key to `getCellInfo()`, which indicates whether the Cell is currently connected to an upstream mirror. (#3041)

17.51.2 Bugfixes

- Fix an issue in the Storm grammar where part of a query could potentially be incorrectly parsed as an unquoted case statement. (#3032)
- Fix an issue where exceptions could be raised which contained data that was not JSON serializable. `$lib.raise` arguments must now also be JSON safe. (#3029)
- Fix an issue where a spawned process returning a non-pickleable exception would not be handled properly. (#3036)
- Fix an issue where a locked user could login to a Synapse service on a TLS Telepath connection if the connection presented a trusted client certificate for the locked user. (#3035)
- Fix a bug in `Scope.enter()` where the added scope frame was not removed when the context manager was exited. (#3021)

- Restoring a service via the `SYN_RESTORE_HTTPS_URL` environment variable could timeout when downloading the file. The total timeout for this process has been disabled. (#3042)

17.51.3 Improved Documentation

- Update the Synapse glossary to add terms related to the permissions system. (#3031)
- Update the model docstrings for the `risk` model. (#3027)

17.51.4 Deprecations

- The `ctor` support in `Scope` has been removed. The population of the global default scope with environment variables has been removed. (#3021)

17.52 v2.123.0 - 2023-02-22

17.52.1 Automatic Migrations

- If the `risk:vuln:cvss:av` property equals `V` it is migrated to `P`. (#3013)
- Parse `inet:http:cookie` nodes to populate the newly added `:name` and `:value` properties. (#3015)
- See *Data Migration* for more information about automatic migrations.

17.52.2 Features and Enhancements

- Added the `belief` model which includes the following new forms: (#3015)

  `belief:system`
  A belief system such as an ideology, philosophy, or religion.

  `belief:tenet`
  A concrete tenet potentially shared by multiple belief systems.

  `belief:subscriber`
  A contact which subscribes to a belief system.

  `belief:system:type:taxonomy`
  A hierarchical taxonomy of belief system types.

- Added declaration for `risk:compromise -(uses)> ou:technique` light-weight edges. (#3015)
- Updated `inet:http:session` and `inet:http:request` forms to include the following property: (#3015)

  `:cookies`
  An array of `inet:http:cookie` values associated with the node.

- Updated the `inet:http:cookie` form to include the following properties: (#3015)

  `name`
  The name of the cookie preceding the equal sign.

  `value`
  The value of the cookie after the equal sign if present.

- Added logic to allow constructing multiple `inet:http:cookie` nodes by automatically splitting on `;` such as `foo=bar; baz=faz` (#3015)
- Updated `it:log:event` to add the following properties: (#3015)

  type
  : An `it:log:event:type:taxonomy` type for the log entry.

  ext:id
  : An external ID that uniquely identifies this log entry.

  product
  : An `it:prod:softver` of the product which produced the log entry.

- Updated the `risk:compromise` form to include the following properties: (#3015)

  goal
  : An `ou:goal` node representing the assessed primary goal of the compromise.

  goals
  : An array of `ou:goal` nodes representing additional goals of the compromise.

- Updated `risk:attack` and `risk:compromise` forms to deprecate the `techniques` property in favor of using `-(uses)> ou:technique` light-weight edges. (#3015)
- Updates to the `inet:dns`, and `media` models. (#3005) (#3017)

  inet:dns:answer
  : Remove all read-only flags present on the secondary properties for this form.

  media:news
  : Add an `updated` property to record last time the news item was updated.

- Updated `inet:flow` to include the following properties: (#3017)

  src:ssh:key
  : The key sent by the client as part of an SSH session setup.

  dst:ssh:key
  : The key sent by the server as part of an SSH session setup.

  src:ssl:cert
  : The x509 certificate sent by the client as part of an SSL/TLS negotiation.

  dst:ssl:cert
  : The x509 certificate sent by the server as part of an SSL/TLS negotiation.

  src:rdp:hostname
  : The hostname sent by the client as part of an RDP session setup.

  src:rdp:keyboard:layout
  : The keyboard layout sent by the client as part of an RDP session setup.

- Add `synapse.utils.stormcov`, a Coverage.py plugin for measuring code coverage of Storm files. (#2961)
- Clean up several references to the `cell.auth` object in HTTP API handlers. Move the logic in `/api/v1/auth/onepass/issue` API handler to the base Cell. (#2998) (#3004)
- Clarify the error message encountered by a Synapse mirrored service if the mirror gets desynchronized from its upstream service. (#3006)
- Update how read-only properties are handled during merges. The `.created` property will always be set when merging a node down. If two nodes have other conflicting read-only property values, those will now emit a warning in the Storm runtime. (#2989)

- The Axon.wget() API response now includes HTTP request history, which is added when the API request encounters redirects. The $lib.axon.wget() Storm API now includes information about the original request URL. This data is now used to create inet:urlredir nodes, such as when the Storm wget command is used to retrieve a file. (#3011)
- Ensure that BadTypeValu exceptions raised when normalizing invalid data with the time type includes the value in the exception message. (#3009)
- Add a callback on Slab size expansion to trigger a free disk space check on the related cell. (#3016)
- Add support for choices in Storm command arguments. (#3019)
- Add an optional parameter to the Storm uniq command to allow specifying a relative property or variable to operate on rather than node iden. (#3018)
- Synapse HTTP API logs now include the user iden and username when that information is available. For deployments with structured logging enabled, the HTTP path, HTTP status code, user iden, and username are added to that log message. (#3007)
- Add web_useriden and web_username attributes to the Synapse HTTP Handler class. These are used for HTTP request logging to populate the user iden and username data. These are automatically set when a user authenticates using a session token or via basic authentication. The HTTP Session tracking now tracks the username at the time the session was created. The _web_user value, which previously pointed to a heavy HiveUser object, is no longer populated by default. (#3007)
- Add $lib.inet.http.codereason Storm API for translating HTTP status codes to reason phrases. inet:http:resp objects now also have a reason value populated. (#3023)
- Update the minimum version of the cryptography library to 39.0.1 and the minimum version of the pyopenssl library to 23.0.0. (#3022)

17.52.3 Bugfixes

- The Storm wget command created inet:urlfile nodes with the url property of the resolved URL from aiohttp. This made it so that a user could not pivot from an inet:url node which had a URL encoded parameter string to the resulting inet:urlfile node. The inet:urlfile nodes are now made with the original request URL to allow that pivoting to occur. (#3011)
- The Axon.wget() and $lib.axon.wget() APIs returned URLs in the url field of their responses which did not contain fragment identifiers. These API responses now include the fragment identifier if it was present in the resolved URL. (#3011)
- The Storm tree command did not properly handle Storm query arguments which were declared as storm:query types. (#3012)
- Remove an unnecessary permission check in the Storm movenodes command which could cause the command to fail. (#3002)
- When a user email address was provided to the HTTP API /api/v1/auth/adduser, the handler did not properly set the email using change controlled APIs, so that information would not be sent to mirrored cells. The email is now being set properly. (#2998)
- The risk:vuln:cvss:av enum incorrectly included V instead of P. (#3013)
- Fix an issue where the ismax specification on time types did not merge time values correctly. (#3017)
- Fix an issue where using a function call to specify the tag in a tagprop operation would not be correctly parsed. (#3020)

17.52.4 Improved Documentation

- Update copyright notice to always include the current year. (#3010)

17.52.5 Deprecations

- The `synapse.lib.httpapi.Handler.user()` and `synapse.lib.httpapi.Handler.getUserBody()` methods are marked as deprecated. These methods will be removed in Synapse `v2.130.0`. (#3007)

17.53 v2.122.0 - 2023-01-27

17.53.1 Features and Enhancements

- Updates to the `biz`, `file`, `lang`, `meta`, `pol`, and `risk` models. (#2984)

  `biz:service`
  Add a `launched` property to record when the operator first made the service available.

  `file:bytes`
  Add `exe:compiler` and `exe:packer` properties to track the software used to compile and encode the file.

  `lang:language`
  Add a new guid form to represent a written or spoken language.

  `lang:name`
  Add a new form to record the name of a language.

  `meta:node`
  Add a `type` property to record the note type.

  `meta:note:type:taxonomy`
  Add a form to record an analyst defined taxonomy of note types.

  `pol:country`
  Correct the `vitals` property type from `ps:vitals` to `pol:vitals`.

  `ps:contact`
  Add a `lang` property to record the language specified for the contact.

  Add a `langs` property to record the alternative languages specified for the contact.

  `ps:skill`
  Add a form to record a specific skill which a person or organization may have.

  `ps:skill:type:taxonomy`
  Add a form to record a taxonomy of skill types.

  `ps:proficiency`
  Add a form to record the assessment that a given contact possesses a specific skill.

  `risk:alert`
  Add a `priority` property that can be used to rank alerts by priority.

  `risk:compromise`
  Add a `severity` property that can be used as a relative severity score for the compromise.

  `risk:threat`
  Add a `type` property to record the type of the threat cluster.

risk:threat:type:taxonomy
Add a form to record a taxonomy of threat types.

- Add support for Python 3.10 to Synapse. (#2962)
- Update the Synapse docker containers to be built from a Debian based image, instead of an Ubuntu based image. These images now use Python 3.10 as the Python runtime. (#2962)
- Add an optional `--type` argument to the Storm `note.add` command. (#2984)
- Add a Storm command, `gen.lang.language`, to lift or generate a `lang:language` node by name. (#2984)
- Update the allowed versions of the `cbor2` library; and upgrade the versions of `aiostmplib` and `aiohttp-socks` to their latest versions. (#2986)
- The `X-XSS-Protection` header was removed from the default HTTP API handlers. This header is non-standard and only supported by Safari browsers. Service deployments which rely on this header should use the `https:headers` configuration option to inject that header into their HTTP responses. (#2997)

17.53.2 Bugfixes

- Malformed hash values normalized as `file:bytes` raised exceptions which were not properly caught, causing Storm ?= syntax to fail. Malformed values are now properly handled in `file:bytes`. (#3000)

17.53.3 Improved Documentation

- Update the Storm filters user guide to include expression filters (#2997)
- Update Storm type-specific behavior user guide to clarify `guid` deconfliction use cases and some associated best practices. (#2997)
- Update Storm command reference user guide to document `gen.*` commands. (#2997)

17.53.4 Deprecations

- The Cortex APIs `provStacks()` and `getProvStack(iden)` have been removed. (#2995)

17.54 v2.121.1 - 2022-01-23

17.54.1 Bugfixes

- When creating Storm Macros using `v2.121.0`, the creator of the Macro was incorrectly set to the `root` user. This is now set to the user that created the macro using the Storm `macro.set` command or the `$lib.macro.set()` API. (#2993)

17.55 v2.121.0 - 2022-01-20

17.55.1 Automatic Migrations

- Storm Macros stored in the Cortex are migrated from the Hive to the Cortex LMDB slab. (#2973)
- See *Data Migration* for more information about automatic migrations.

17.55.2 Features and Enhancements

- Updates to the `inet` and `ou` models. (#2982) (#2987)

  `inet:dns:soa`
  The `fqdn`, `ns` and `email` properties had the read-only flag removed from them.

  `ou:org`
  Add a `goals` property to record the assessed goals of the organization.

- Add extended permissions for Storm Macro functionality using a new simplified permissions system. This allows users to opt into assigning users or roles the permission to read, write, administrate, or deny access to their Macros. These permissions can be set by the Storm `$lib.macro.grant()` API. (#2973)
- Add extended information about a Storm Macro, including its creation time, update time, and a description. The Macro name, description and Storm can now be set via the Storm `$lib.macro.mod()` API. (#2973)
- Allow users and Power-Ups to store graph projection definitions in the Cortex. Graph projections have the same simplified permissions system applied to them as introduced for Storm Macros. Storm users can now also load a stored graph projection into a running Storm query. These new features are exposed via the Storm `$lib.graph` APIs. (#2914)
- The disk space required to make the backup of a Synapse service is now checked prior to a live backup being made. If there is insufficient storage to make the backup on the volume storing the backup, a LowSpace exception will be raised. (#2990)

17.55.3 Bugfixes

- When normalizing the `inet:email` type, an unclear Python `ValueError` could have been raised to a user. This is now caught and a specific `BadTypeValu` exception is raised. (#2982)
- The `synapse.exc.StormRaise` exception caused an error when recreating the exception on the client side of a Telepath connection. This exception will now raise properly on the caller side. (#2985)
- When using the Storm `diff` command to examine a forked View, if a node was deleted out from the base layer and edited in the fork, an exception would be raised. This situation is now properly handled. (#2988)

17.55.4 Improved Documentation

- Update the Storm User Guide section on variables for clarity. (#2968)
- Correct Provenance API deprecation notice from `v2.221.0` to `v2.122.0`. (#2981)

17.56 v2.120.0 - 2023-01-11

17.56.1 Features and Enhancements

- Update to the `risk` models. (#2978)

  `risk:threat`
  Add a `merge:time` and `merged:isnow` properties to track when a threat cluster was merged with another threat cluster.

  `risk:alert`
  Add an `engine` property to track the software engine that generated the alert.

- Add events for `trigger:add`, `trigger:del`, and `trigger:set` to the Beholder API. (#2975)

17.56.2 Bugfixes

- Fix an infinite loop in `synapse.tools.storm` when using the tool in an environment without write access to the history file. (#2977)

17.57 v2.119.0 - 2023-01-09

17.57.1 Features and Enhancements

- Updates to the `biz`, `econ`, `ou`, and `risk` models. (#2931)

  `biz:listing`
  Add a form to track a specific product or service listed for sale at a given price by a specific seller.

  `biz:service`
  Add a form to track a service performed by a specific organization.

  `biz:service:type`
  Add a form to record an analyst defined taxonomy of business services.

  `biz:bundle`
  Add a `service` property to record the service included in the bundle.

  Deprecate the `deal` and `purchase` secondary properties in favor of `econ:receipt:item` to represent bundles being sold.

  `biz:product`
  Add a `price:currency` property to denote the currency of the prices.

  Add a `maker` property to represent the contact information for the maker of a product.

  Deprecate the `madeby:org`, `madeby:orgname`, `madeby:orgfqdn` properties in favor of using the new `maker` property.

  `econ:receipt:item`
  Add a form to represent a line item included as part of a purchase.

  `econ:acquired`
  Deprecate the form in favor of an `acquired` light edge.

`ou:campaign`
Add a `budget` property to record the budget allocated for the campaign.

Add a `currency` property to record the currency of the `econ:price` secondary properties.

Add a `result:revenue` property to record the revenue resulting from the campaign.

Add a `result:pop` property to record the count of people affected by the campaign.

`risk:alert:verdict:taxonomy`
Add a form to record an analyst defined taxonomy of the origin and validity of an alert.

`risk:alert`
Add a `benign` property to record if the alert has been confirmed as benign or malicious.

Add a `verdict` property to record the analyst verdict taxonomy about why an alert is marked as benign or malicious.

- Annotate the following light edges. (#2931)

`acquired`
When used with an `econ:purchase` node, the edge indicates the purchase was used to acquire the target node.

`ipwhois`
When used with an `inet:whois:iprec` node and `inet:ipv4` or `inet:ipv6` nodes, the edge indicates the source IP whois record describes the target IP address.

- Add a new Cell configuration option, `limit:disk:free`. This represents the minimum percentage of free disk space on the volume hosting a Synapse service that is required in order to start up. This value is also monitored every minute and will disable the Cell Nexus if the free space drops below the specified value. This value defaults to five percent (`5 %`) free disk space. (#2920)

17.57.2 Improved Documentation

- Add a Devops task related to configuration of the free space requirement. (#2920)

17.58 v2.118.0 - 2023-01-06

17.58.1 Features and Enhancements

- Updates to the `inet`, `pol`, and `ps` models. (#2970) (#2971)

`inet:tunnel`
Add a form to represent the specific sequence of hosts forwarding connections, such as a VPN or proxy.

`inet:tunnel:type:taxonomy`
Add a form to record an analyst defined taxonomy of network tunnel types.

`pol:country`
Add a `government` property to represent the organization for the government of the country.

`ps:contact`
Add a `type` property to record the taxonomy of the node. This may be used for entity resolution.

`ps:contact:type:taxonomy`
Add a form to record an analyst defined taxonomy of contact types.

- Add the following Storm commands to help with analyst generation of several guid node types: (#2970)

`gen.it.prod.soft`
: Lift (or create) an `it:prod:soft` node based on the software name.

`gen.ou.industry`
: Lift (or create) an `ou:industry` node based on the industry name.

`gen.ou.org`
: Lift (or create) an `ou:org` node based on the organization name.

`gen.ou.org.hq`
: Lift (or create) the primary `ps:contact` node for the ou:org based on the organization name.

`gen.pol.country`
: Lift (or create) a `pol:country` node based on the 2 letter ISO-3166 country code.

`gen.pol.country.government`
: Lift (or create) the `ou:org` node representing a country's government based on the 2 letter ISO-3166 country code.

`gen.ps.contact.email`
: Lift (or create) the `ps:contact` node by deconflicting the email and type.

`gen.risk.threat`
: Lift (or create) a `risk:threat` node based on the threat name and reporter name.

`gen.risk.tool.software`
: Lift (or create) a `risk:tool:software` node based on the tool name and reporter name.

`gen.risk.vuln`
: Lift (or create) a `risk:vuln` node based on the CVE.

- Add `$lib.gen.riskThreat()`, `$lib.gen.riskToolSoftware()`, `$lib.gen.psContactByEmail()`, and `$lib.gen.polCountryByIso2()` Storm API functions to assist in generating `risk:threat`, `risk:tool:software`, `ps:contact` and `pol:country` nodes. (#2970)
- Update the CRL bundled within Synapse to revoke the `The Vertex Project Code Signer 00` key. (#2972)

17.58.2 Bugfixes

- Fix an issue in the Axon `csvrows()` and `readlines()` APIs which could cause the Axon service to hang. (#2969)

17.59 v2.117.0 - 2023-01-04

17.59.1 Automatic Migrations

- The `risk:tool:software:soft:names` and `risk:tool:software:techniques` properties are migrated to being unique arrays. (#2950)
- See *Data Migration* for more information about automatic migrations.

17.59.2 Features and Enhancements

- Updates to the `risk` model. (#2950)

  `risk:tool:software`
  The `soft:names` and `techniques` properties are converted into sorted and uniqued arrays.

- Add support to the Cortex `addStormPkg()` and `$lib.pkg.add()` APIs to load Storm Packages which have been signed to allow cryptographic signature verification. Root CA and intermediate CA certificates have been embedded into Synapse to allow for verification of Rapid Power-Ups signed by The Vertex Project. (#2940) (#2957) (#2963)
- Update `synapse.tools.genpkg` to add optional code signing to Storm packages that it creates. (#2940)
- Update `synapse.tools.genpkg` to require the packages it produces will be JSON compatible when serialized, to avoid possible type coercion issues introduced by the Python `json` library. (#2958)
- Update `synapse.tools.easycert` to allow for creating code signing certificates and managing certificate revocation lists (CRLs). (#2940)
- Add the Nexus index (`nexsindx`) value to the data returned by the `getCellInfo()` APIs. (#2949)
- Allow the Storm backtick format strings to work with multiline strings. (#2956)
- The Storm `Bytes.json()` method now raises exceptions that are `SynErr` subclasses when encountering errors. This method has been updated to add optional `encoding` and `errors` arguments, to control how data is deserialized. (#2945)
- Add support for registering an OAuth2 provider in the Cortex and having user tokens automatically refreshed in the background. These APIs are exposed in Storm under the `$lib.inet.http.oauth.v2` library. (#2910)
- STIX validation no longer caches any downloaded files it may use when attempting to validate STIX objects. (#2966)
- Modified the behavior of Storm emitter functions to remove the read-ahead behavior. (#2953)

17.59.3 Bugfixes

- Fix some error messages in the Snap which did not properly add variables to the message. (#2951)
- Fix an error in the `synapse.tools.aha.enroll` command example. (#2948)
- Fix an error with the `merge` command creating `No form named None` warnings in the Cortex logs. (#2952)
- Fix the Storm `inet:smtp:message` getter and setter for the `html` property so it will correctly produce HTML formatted messages. (#2955)
- Several `certdir` APIs previously allowed through `openssl.crypto.X509StoreContextError` and `openssl.crypto.Error` exceptions. These now raise Synapse `BadCertVerify` and `BadCertBytes` exceptions. (#2940)
- Fix an issue where a Storm package's `modconf` values were mutable. (#2964)

17.59.4 Improved Documentation

- Removed outdated Kubernetes related devops documentation as it is in the process of being rewritten. (#2948)

17.59.5 Deprecations

- The Cortex APIs `provStacks()` and `getProvStack(iden)` and the corresponding Cortex configuration option `provenance:en` have been marked as deprecated and are planned to be removed in `v2.122.0`. (#2682)

17.60 v2.116.0 - 2022-12-14

17.60.1 Automatic Migrations

- The `ou:contract:award:price` and `ou:contract:budget:price` properties are migrated from `econ:currency` to `econ:price` types. (#2943)
- See *Data Migration* for more information about automatic migrations.

17.60.2 Features and Enhancements

- Updates to the `ou` model. (#2943)

  `ou:contract`
  : The `award:price` and `budget:price` properties had their types changed from `econ:currency` to `econ:price`. Add a `currency` secondary property to record the currency of the `econ:price` values.

17.60.3 Bugfixes

- The `synapse.tools.genpkg` tool could raise a Python `TypeError` when the specified package file did not exist. It now raises a `NoSuchFile` exception. (#2941)
- When a service is provisioned with an `aha:provision` URL placed in a `cell.yaml` file, that could create an issue when a mirror is deployed from that service, preventing it from starting up a second time. Services now remove the `aha:provision` key from a `cell.yaml` file when they are booted from a mirror if the URL does not match the boot URL. (#2939)
- When deleting a node from the Cortex, secondary properties defined as arrays were not checked for their references to other nodes. These references are now properly checked prior to node deletion. (#2942)

17.60.4 Improved Documentation

- Add a Devops task for stamping custom users into Synapse containers to run services with arbitrary user and group id values. (#2921)
- Remove an invalid reference to `insecure` mode in HTTP API documentation. (#2938)

17.61 v2.115.1 - 2022-12-02

17.61.1 Features and Enhancements

- Patch release to include an updated version of the `pytest` library in containers.

17.62 v2.115.0 - 2022-12-01

17.62.1 Automatic Migrations

- The `inet:flow:dst:softnames` and `inet:flow:dst:softnames` properties are migrated from `it:dev:str` to `it:prod:softname` types. (#2930)
- See *Data Migration* for more information about automatic migrations.

17.62.2 Features and Enhancements

- Updates to the `inet` model. (#2930)

  `inet:flow`
  The `dst:softnames` and `src:softnames` properties had their types changed from `it:dev:str` values to `it:prod:softname`.

- Add support for secondary property pivots where the target property is an array type. (#2922)
- The Storm API `$lib.bytes.has()` now returns a false value when the input is null. (#2924)
- When unpacking loop values in Storm, use the primitive value when the item being unpacked is a Storm primitive. (#2928)
- Add a `--del` option to the `synapse.tools.moduser` tool to allow removing a user from a service. (#2933)
- Add entrypoint hooks to the Aha, Axon, Cortex, Cryotank, and JsonStor containers that allow a user to hook the container boot process. (#2919)
- Temporary files created by the Axon, Cortex and base Cell class are now created in the cell local `tmp` directory. In many deployments, this would be located in `/vertex/storage/tmp`. (#2925)
- Update the allowed versions of the `cbor2` and `pycryptodome` libraries. For users installing `synapse[dev]`, `coverage`, `pytest`, `pytest-cov` and `pytest-xdist` are also updated to their latest versions. (#2935)

17.62.3 Bugfixes

- When a Storm Dmon definition lacked a `view` iden, it would previously default to using the Cortex default view. Dmons now prefer to use the user default view before using the Cortex default view. This situation would only happen with Dmons created via the Telepath API where the `view` iden was not provided in the Dmon definition. (#2929)
- Non-integer mask values provided to `inet:cidr4` types now raise a `BadTypeValu` exception. (#2932)
- Fix an incorrect call to `os.unlink` in `synapse.tools.aha.enroll`. (#2926)

17.62.4 Improved Documentation

- Update the automation section of the Synapse User guide, expanding upon the use of cron jobs and triggers across views and forks. (#2917)

17.63 v2.114.0 - 2022-11-15

17.63.1 Features and Enhancements

- Updates to the `crypto` model. (#2909)

  `crypto:key`
  Add `iv` and `mode` properties to record initialization vectors and cipher modes used with a key.

- Allow the creator for Cron jobs and the user for Triggers to be set. This can be used to effectively change the ownership of these automation elements. (#2908)
- When Storm package `onload` queries produce print, warning, or error messages, those now have the package name included in the message that is logged. (#2913)
- Update the Storm package schema to allow declaring configuration variables. (#2880)

17.63.2 Bugfixes

- The `delCertPath()` APIs in `synapse.lib.easycert` no longer attempt to create a file path on disk when removing the reference count to a certificate path. (#2907)
- Fix error handling when Axon is streaming files with the `readlines()` and `csvrows()` APIs. (#2911)
- The Storm `trigger.list` command failed to print triggers which were created in a Cortex prior to `v2.71.0`. These triggers no longer generate an exception when listed. (#2915)
- Fix an error in the HTTP API example documentation for the `requests` example. (#2918)

17.63.3 Improved Documentation

- Add a Devops task to enable the Python warnings filter to log the use of deprecated Synapse APIs. Python APIs which have been deprecated have had their docstrings updated to reflect their deprecation status. (#2905)

17.64 v2.113.0 - 2022-11-04

17.64.1 Automatic Migrations

- The `risk:tool:software:type` property is migrated to the `risk:tool:software:taxonomy` type. (#2900)
- See *Data Migration* for more information about automatic migrations.

17.64.2 Features and Enhancements

- Updates to the `inet`, `infotech`, `media`, `meta`, `ou`, and `risk` models. (#2897) (#2900) (#2903)

  `inet:email:message:link`
  Add a `text` property to record the displayed hypertext link if it was not a raw URL.

  `inet:web:acct`
  Add a `banner` property representing the banner image for the account.

  `inet:web:mesg`
  Add a `deleted` property to mark if a message was deleted.

  `inet:web:post:link`
  Add a form to record a link contained in the post text.

  `it:mitre:attack:group`
  Add an `isnow` property to record the potential for MITRE groups to be deprecated and renamed.

  `it:mitre:attack:software`
  Add an `isnow` property to record the potential for MITRE software to be deprecated and renamed.

  `it:prod:soft:taxonomy`
  Add a form to record an analyst defined taxonomy of software.

  `it:prod:soft`
  Add a `type` property to record the taxonomy of the software. Deprecated the `techniques` property in favor of the `uses` light edge.

  `it:sec:cve`
  Deprecated the `desc`, `url` and `references` properties in favor of using the `risk:vuln:cve:desc`, `risk:vuln:cve:url`, and `risk:vuln:cve:references` properties.

  `media:news`
  Add a `topics` array property to record a list of relevant topics in the article.

  `media:topic`
  Add a form for recording different media topics.

  `meta:rule`
  Add a `url` property to record a URL that documents as rule.

  Add an `ext:id` property to record an external identifier for the rule.

  `meta:sophistication`
  Add a form to record sophistication score with named values: `very low`, `low`, `medium`, `high`, and `very high`.

  `ou:campaign`
  Add a `sophistication` property to record the assessed sophistication of a campaign.

  Deprecate the `techniques` property in favor of using the `uses` light edge.

  `ou:hasgoal`
  Deprecate the `ou:hasgoal` form in favor of using the `ou:org:goals` property.

  `ou:org`
  Deprecate the `techniques` property in favor of using the `uses` light edge.

  `ou:technique`
  Add a `sophistication` property to record the assessed sophistication of a technique.

  `risk:alert`
  Add a `url` property for a URL that documents the alert.

Add an `ext:id` property to record an external ID for the alert.

risk:attack
Add a `sophistication` property to record the assessed sophistication of an attack.

risk:availability
Add a taxonomy for availability status values.

risk:threat
Add a `sophistication` property to record the assessed sophistication of a threat cluster.

Deprecate the `techniques` property in favor of the `uses` light edge.

risk:tool:software
Add an `availability` property to record the assessed availability of the tool.

Add a `sophistication` property to record the assessed sophistication of the software.

Migrate the `type` property to `risk:tool:software:taxonomy`.

Deprecate the `techniques` property in favor of the `uses` light edge.

risk:tool:software:taxonomy
Rename the type `risk:tool:taxonomy` to `risk:tool:software:taxonomy`.

risk:vuln
Add a `mitigated` property to record if a mitigation or fix is available for the vulnerability.

Add an `exploited` property to record if the vulnerability has been exploited in the wild.

Add `timeline:discovered`, `timeline:published`, `timeline:vendor:notified`, `timeline:vendor:fixed`, and `timeline:exploited` properties to record the timeline for significant events on a vulnerability.

Add `cve:desc`, `cve:url`, and `cve:references` secondary properties to record information about the CVE associated with a vulnerability.

Add `` `nist:nvd:source `` to record the name of the organization which reported the vulnerability in the NVD.

Add `nist:nvd:published` and `nist:nvd:modified` to record when the vulnerability was first published, and later modified, in the NVD.

Add `cisa:kev:name`, `cisa:kev:desc`, `cisa:kev:action`, `cisa:kev:vendor`, `cisa:kev:product`, `cisa:kev:added`, `cisa:kev:duedate` properties to record information about the CISA KEV database entry for the vulnerability.

- Annotate the following light edges. (#2900)

seen
When used with `meta:source` nodes, the edge indicates the target node was observed by the source node.

stole
When used with a `risk:compromise` node, the edge indicates the target node was stolen or copied as a result of the compromise.

targets
When used with `risk:attack`, the edge indicates the target node is targeted by the attack.

When used with `risk:attack` and `ou:industry` nodes, the edge indicates the attack targeted the industry

When used with `risk:threat`, the edge indicates the target node is targeted by the threat cluster.

When used with `risk:threat` and `ou:industry` nodes, the edge indicates the threat cluster targets the industry.

uses

When used with `ou:campaign` and `ou:technique` nodes, the edge indicates the campaign used a given technique.

When used with `ou:org` and `ou:technique` nodes, the edge indicates the organization used a given technique.

When used with `risk:threat`, the edge indicates the target node was used to facilitate the attack.

When used with `risk:attack` and `ou:technique` nodes, the edge indicates the attack used a given technique.

When used with `risk:attack` and `risk:vuln` nodes, the edge indicates the attack used the vulnerability.

When used with `risk:tool:software`, the edge indicates the target node is used by the tool.

When used with `risk:tool:software` and `ou:technique` nodes, the edge indicates the tool uses the technique.

When used with `risk:tool:software` and `risk:vuln` nodes, the edge indicates the tool used the vulnerability.

When used with `risk:threat`, the edge indicates the target node was used by threat cluster.

When used with `risk:threat` and `ou:technique` nodes, the edge indicates the threat cluster uses the technique.

When used with `risk:threat` and `risk:vuln` nodes, the edge indicates the threat cluster uses the vulnerability.

- Add `$lib.gen.vulnByCve()` to help generate `risk:vuln` nodes for CVEs. (#2903)
- Add a unary negation operator to Storm expression syntax. (#2886)
- Add `$lib.crypto.hmac.digest()` to compute RFC2104 digests in Storm. (#2902)
- Update the Storm `inet:http:resp.json()` method to add optional `encoding` and `errors` arguments, to control how data is deserialized. (#2898)
- Update the Storm `bytes.decode()` method to add an optional `errors` argument, to control how errors are handled when decoding data. (#2898)
- Logging of role and user permission changes now includes the authgate iden for the changes. (#2891)

17.64.3 Bugfixes

- Catch `RecursionError` exceptions that can occur in very deep Storm pipelines. (#2890)

17.64.4 Improved Documentation

- Update the Storm reference guide to explain backtick format strings. (#2899)
- Update `guid` section on Storm type-specific behavior doc with some additional guid generation examples. (#2901)
- Update Storm control flow documentation to include `init`, `fini`, and `try` / `catch` examples. (#2901)
- Add examples for creating extended model forms and properties to the Synapse admin guide. (#2904)

17.65 v2.112.0 - 2022-10-18

17.65.1 Features and Enhancements

- Add `--email` as an argument to `synapse.tools.moduser` to allow setting a user's email address. (#2891)
- Add support for `hxxp[s]:` prefixes in scrape functions. (#2887)
- Make the SYNDEV_NEXUS_REPLAY resolution use `s_common.envbool()` in the `SynTest.withNexusReplay()` helper. Add `withNexusReplay()` calls to all test helpers which make Cells which previously did not have it available. (#2889) (#2890)
- Add implementations of `getPermDef()` and `getPermDefs()` to the base Cell class. (#2888)

17.65.2 Bugfixes

- Fix an idempotency issue in the JsonStor multiqueue implementation. (#2890)

17.65.3 Improved Documentation

- Add Synapse-GCS (Google Cloud Storage) Advanced Power-Up to the Power-Ups list.

17.66 v2.111.0 - 2022-10-12

17.66.1 Features and Enhancements

- Update the Storm grammar to allow specifying a tag property with a variable. (#2881)
- Add log messages for user and role management activities in the Cell. (#2877)
- The logging of service provisioning steps on Aha and when services were starting up was previously done at the `DEBUG` level. These are now done at the `INFO` level. (#2883)
- The `vertexproject/synapse:` docker images now have the environment variable `SYN_LOG_LEVEL` set to `INFO`. Previously this was `WARNING`. (#2883)

17.66.2 Bugfixes

- Move the Nexus `runMirrorLoop` task to hang off of the Telepath Proxy and not the Telepath client. This results in a faster teardown of the `runMirrorLoop` task during Nexus shutdown. (#2878)
- Remove duplicate tokens presented to users in Storm syntax errors. (#2879)
- When bootstrapping a service mirror with Aha provisioning, the `prov.done` file that was left in the service storage directory was the value from the upstream service, and not the service that has been provisioned. This resulted in `NoSuchName` exceptions when restarting mirrors. The bootstrapping process now records the correct value in the `prov.done` file. (#2882)

17.67 v2.110.0 - 2022-10-07

17.67.1 Features and Enhancements

- Updates to the geo model. (#2872)

  geo:telem
  Add an accuracy property to record the accuracy of the telemetry reading.

- Add Nexus support to the Axon, to enable mirrored Axon deployments. (#2871)
- Add Nexus support for HTTP API sessions. (#2869)
- Add support for runtime string formatting in Storm. This is done with backtick (` ) encapsulated strings. An example of this is `$world='world' $lib.print(`hello {$world}`)` (#2870) (#2875)
- Expose user profile storage on the auth:user object, with the profile ctor. (#2876)
- Storm package command names are now validated against the same regex used by the grammar. The synapse.tools.genpkg tool now validates the compiled package against the same schema used by the Cortex. (#2864)
- Add $lib.gen.newsByUrl() and $lib.gen.softByName() to help generate media:news and it:prod:soft nodes, respectively. (#2866)
- Add a new realtime event stream system to the Cell, accessible remotely via CellApi.behold() and a websocket endpoint, /api/v1/behold. This can be used to get realtime changes about services, such as user creation or modification events; or layer and view change events in the Cortex. (#2851)
- Update stored user password hashing to use PBKDF2. Passwords are migrated to this format as successful user logins are performed. (#2868)
- Add the ability to restore a backup tarball from a URL to the Cell startup process. When a Cell starts via initFromArgv(), if the environment variable SYN_RESTORE_HTTPS_URL is present, that value will be used to retrieve a tarball via HTTPS and extract it to the service local storage, removing any existing data in the directory. This is done prior to any Aha based provisioning. (#2859)

17.67.2 Bugfixes

- The embedded Axon inside of a Cortex (used when the axon config option is not set) did not properly have its cell parent set to the Cortex. This has been corrected. (#2857)
- Fix a typo in the cron.move help. (#2858)

17.67.3 Improved Documentation

- Update Storm and Storm HTTP API documentation to show the set of opts and different types of message that may be streamed by from Storm APIs. Add example HTTP API client code to the Synapse repository. (#2834)
- Update the Data Model and Analytical model background documentation. Expand on the discussion of light edges use. Expand discussion of tags versus forms, linking the two via :tag props. (#2848)

17.67.4 Deprecations

- The Cortex HTTP API endpoint `/api/v1/storm/nodes` has been marked as deprecated. (#2682)
- Add deprecation notes to the help for the Storm `splice.undo` and `splice.list` commands. (#2861)
- Provisional Telepath support for Consul based lookups was removed. (#2873)

17.68 v2.109.0 - 2022-09-27

17.68.1 Features and Enhancements

- Add a `format()` API to `str` variables in Storm. (#2849)
- Update the Telepath user resolution for TLS links to prefer resolving users by the Cell `aha:network` over the certificate common name. (#2850)
- Update all Synapse tools which make telepath connections to use the `withTeleEnv()` helper. (#2844)
- Update the Telepath and HTTPs TLS listeners to drop RSA based key exchanges and disable client initiated renegotiation. (#2845)
- Update the minimum allowed versions of the `aioimaplib` and `oauthlib` libraries. (#2847) (#2854)

17.68.2 Bugfixes

- Correct default Telepath `cell://` paths in Synapse tools. (#2853)
- Fix typos in the inline documentation for several model elements. (#2852)
- Adjust expression syntax rules in Storm grammar to remove incorrect whitespace sensitivity in certain expression operators. (#2846)

17.68.3 Improved Documentation

- Update Storm and Storm HTTP API documentation to show the set of `opts` and different types of message that may be streamed by from Storm APIs. Add example HTTP API client code to the Synapse repository. (#2834)
- Update the Data Model and Analytical model background documentation. Expand on the discussion of light edges use. Expand discussion of tags versus forms, linking the two via `:tag` props. (#2848)

17.69 v2.108.0 - 2022-09-12

17.69.1 Features and Enhancements

- Update the Telepath TLS connections to require a minimum TLS version of 1.2. (#2833)
- Update the Axon implementation to use the `initServiceStorage()` and `initServiceRuntime()` methods, instead of overriding `__anit__`. (#2837)
- Update the minimum allowed versions of the `aiosmtplib` and `regex` libraries. (#2832) (#2841)

17.69.2 Bugfixes

- Catch `LarkError` exceptions in all Storm query parsing modes. (#2840)
- Catch `FileNotFound` errors in `synapse.tools.healthcheck`. This could be caused by the tool running during container startup, and prior to a service making its Unix listening socket available. (#2836)
- Fix an issue in `Axon.csvrows()` where invalid data would cause processing of a file to stop. (#2835)
- Address a deprecation warning in the Synapse codebase. (#2842)
- Correct the type of `syn:splice:splice` to be `data`. Previously it was `str`. (#2839)

17.69.3 Improved Documentation

- Replace `livenessProbe` references with `readinessProbe` in the Kubernetes documentation and examples. The `startupProbe.failureThreshold` value was increased to its maximum value. (#2838)
- Fix a typo in the Rapid Power-Up documentation. (#2831)

17.70 v2.107.0 - 2022-09-01

17.70.1 Automatic Migrations

- Migrate the `risk:alert:type` property to a `taxonomy` type and create new nodes as needed. (#2828)
- Migrate the `pol:country:name` property to a `geo:name` type and create new nodes as needed. (#2828)
- See *Data Migration* for more information about automatic migrations.

17.70.2 Features and Enhancements

- Updates to the `geo`, `inet`, `media`, `pol`, `proj`, and `risk` models. (#2828) (#2829)

  `geo:area`
  Add a new type to record the size of a geographic area.

  `geo:place:taxonomy`
  Add a form to record an analyst defined taxonomy of different places.

  `geo:place`
  Add a `type` property to record the taxonomy of a place.

  `inet:web:memb`
  This form has been deprecated.

  `inet:web:member`
  Add a guid form that represents a web account's membership in a channel or group.

  `media:news:taxonomy`
  Add a form to record an analyst defined taxonomy of different types or sources of news.

  `media:news`
  Add a `type` property to record the taxonomy of the news. Add an `ext:id` property to record an external identifier provided by a publisher.

  `pol:vitals`
  Add a guid form to record the vitals for a country.

pol:country
Add names, place, dissolved and vitals secondary properties. The name is changed from a str to a geo:name type. Deprecate the pop secondary property.

pol:candidate
Add an incumbent property to note if the candidate was an incumbent in a race.

proj
Add missing docstrings to the proj model forms.

risk:alert:taxonomy
Add a form to record an analyst defined taxonomy of alert types.

risk:alert
The type property is changed from a str to the risk:alert:taxonomy type.

- Add ** as a power operator for Storm expression syntax. (#2827)
- Add a new test helper, synapse.test.utils.StormPkgTest to assist with testing Rapid Power-Ups. (#2819)
- Add $lib.axon.metrics() to get the metrics from the Axon that the Cortex is connected to. (#2818)
- Add pack() methods to the auth:user and auth:role objects. This API returns the definitions of the User and Role objects. (#2823)
- Change the Storm Package require values to log debug messages instead of raising exceptions if the requirements are not met. Add a $lib.pkg.deps() API that allows inspecting if a package has its dependencies met or has conflicts. (#2820)

17.70.3 Bugfixes

- Prevent None objects from being normalized as tag parts from variables in Storm. (#2822)
- Avoid intermediate conversion to floats during storage operations related to Synapse Number objects in Storm. (#2825)

17.70.4 Improved Documentation

- Add Developer documentation for writing Rapid Power-Ups. (#2803)
- Add the synapse.tests.utils package to the Synapse API autodocs. (#2819)
- Update Devops documentation to note the storage requirements for taking backups of Synapse services. (#2824)
- Update the Storm min and max command help to clarify their usage. (#2826)

17.71 v2.106.0 - 2022-08-23

17.71.1 Features and Enhancements

- Add a new tool, synapse.tools.axon2axon, for copying the data from one Axon to another Axon. (#2813) (#2816)

17.71.2 Bugfixes

- Subquery filters did not update runtime variables in the outer scope. This behavior has been updated to make subquery filter behavior consistent with regular subqueries. (#2815)
- Fix an issue with converting the Number Storm primitive into its Python primitive. (#2811)

17.72 v2.105.0 - 2022-08-19

17.72.1 Features and Enhancements

- Add a Number primitive to Storm to facilitate fixed point math operations. Values in expressions which are parsed as floating point values will now be Numbers by default. Values can also be cast to Numbers with `$lib.math.number()`. (#2762)
- Add `$lib.basex.encode()` and `$lib.basex.decode()` for encoding and decoding strings using arbitrary charsets. (#2807)
- The tag removal operator (`-#`) now accepts lists of tags to remove. (#2808)
- Add a `$node.difftags()` API to calculate and optionally apply the difference between a list of tags and those present on a node. (#2808)
- Scraped Ethereum addresses are now returned in their EIP55 checksummed form. This change also applies to lookup mode. (#2809)
- Updates to the `mat`, `ps`, and `risk` models. (#2804)

  `mass`
  Add a type for storing mass with grams as a base unit.

  `ps:vitals`
  Add a form to record statistics and demographic data about a person or contact.

  `ps:person`
  Add a `vitals` secondary property to record the most recent known vitals for the person.

  `ps:contact`
  Add a `vitals` secondary property to record the most recent known vitals for the contact.

  `risk:tool:taxonomy`
  Add a form to record an analyst defined taxonomy of different tools.

  `risk:tool:software`
  Add a form to record software tools used in threat activity.

  `risk:threat`
  Add `reporter`, `reporter:name`, `org:loc`, `org:names`, and `goals` secondary properties.

- Annotate the following light edges. (#2804)

  `uses`
  When used with `risk:threat` nodes, the edge indicates the target node is used by the source node.

17.72.2 Bugfixes

- Fix language used in the `model.deprecated.check` command. (#2806)
- Remove the `-y` switch in the `count` command. (#2806)

17.73 v2.104.0 - 2022-08-09

17.73.1 Automatic Migrations

- Migrate *crypto:x509:cert:serial* from *str* to *hex* type. Existing values which cannot be converted as integers or hex values will be moved into nodedata under the key `migration:0_2_10` as `{'serial': value}` (#2789)
- Migrate `ps:contact:title` to the `ou:jobtitle` type and create `ou:jobtitle` nodes. (#2789)
- Correct hugenum property index values for values with more than 28 digits of precision. (#2766)
- See *Data Migration* for more information about automatic migrations.

17.73.2 Features and Enhancements

- Updates to the `crypto` and `ps` models. (#2789)

  `crypto:x509:cert`
  The `serial` secondary property has been changed from a `str` to a `hex` type.

  `ps:contact`
  The type of the `title` secondary property has been changed from a `str` to an `ou:jobtitle`.

- Add `$lib.hex.toint()`, `$lib.hex.fromint()`, `$lib.hex.trimext()` and `$lib.hex.signext()` Storm APIs for handling hex encoded integers. (#2789)
- Add `set()` and `setdefault()` APIs on the SynErr exception class. Improve support for unpickling SynErr exceptions. (#2797)
- Add logging configuration to methods which are called in spawned processes, and log exceptions occurring in the processes before tearing them down. (#2795)

17.73.3 Bugfixes

- BadTypeValu errors raised when normalizing a tag timestamp now include the name of the tag being set. (#2797)
- Correct a CI issue that prevented the v2.103.0 Docker images from being published. (#2798)

17.73.4 Improved Documentation

- Update data model documentation. (#2796)

17.74 v2.103.0 - 2022-08-05

17.74.1 Features and Enhancements

- Updates to the `it`, `ou`, and `risk` models. (#2778)

  `it:prod:soft`
  Add a `techniques` secondary property to record techniques employed by the author of the software.

  `ou:campaign`
  Add a `techniques` secondary property to record techniques employed by the campaign.

  `ou:org`
  Add a `techniques` secondary property to record techniques employed by the org.

  `ou:technique`
  Add a form to record specific techniques used to achieve a goal.

  `ou:technique:taxonomy`
  Add a form to record an analyst defined taxonomy of different techniques.

  `risk:attack`
  Add a `techniques` secondary property to record techniques employed during the attack. Deprecate the following secondary properties, in favor of using light edges.

  - `target`
  - `target:host`
  - `target:org`
  - `target:person`
  - `target:place`
  - `used:email`
  - `used:file`
  - `used:host`
  - `used:server`
  - `used:software`
  - `used:url`
  - `used:vuln`
  - `via:email`
  - `via:ipv4`
  - `via:ipv6`
  - `via:phone`

  `risk:compromise`
  Add a `techniques` secondary property to record techniques employed during the compromise.

  `risk:threat`
  Add a form to record a threat cluster or subgraph of threat activity attributable to one group.

- Annotate the following light edges. (#2778)

targets
When used with `ou:org`, `ou:campaign`, `risk:threat`, or `risk:attack` nodes, the edge indicates the target node was targeted by the source node.

uses
When used with an `ou:campaign` or `risk:attack` node, the edge indicates the target node is used by the source node.

- Change the behavior of the Storm `count` command to consume nodes. If the previous behavior is desired, use the `--yield` option when invoking the `count` command. (#2779)
- Add `$lib.random.int()` API to Storm for generating random integers. (#2783)
- Add a new tool, `synapse.tools.livebackup` for taking a live backup of a service. (#2788)
- The Storm `$lib.jsonstor.cacheset()` API now returns a dict containing the path and time. The `$lib.jsonstor.cacheget()` API now has an argument to retrieve the entire set of enveloped data. (#2790)
- Add a HTTP 404 handler for the Axon `v1/by/sha256/<sha256>` endpoint which catches invalid `<sha256>` values. (#2780)
- Add helper scripts for doing bulk Synapse Docker image builds and testing. (#2716)
- Add `aha:\\` support to `synapse.tools.csvtool`. (#2791)

17.74.2 Bugfixes

- Ensure that errors that occur when backing up a service are logged prior to tearing down the subprocess performing the backup. (#2781)
- Add missing docstring for `$lib.stix.import`. (#2786)
- Allow setting tags on a Node from a Storm `List` object. (#2782)

17.74.3 Improved Documentation

- Remove `synapse-google-ct` from the list of Rapid Power-Ups. (#2779)
- Add developer documentation for building Synapse Docker containers. (#2716)
- Fix spelling errors in model documentation. (#2782)

17.74.4 Deprecations

- The `vertexproject/synapse:master-py37` and `vertexproject/synapse:v2.x.x-py37` Docker containers are no longer being built. (#2716)

17.75 v2.102.0 - 2022-07-25

17.75.1 Features and Enhancements

- Updates to the `crypto`, `geo`, `inet`, `mat`, `media`, `ou`, `pol`, and `proj` models. (#2757) (#2771)

  `crypto:key`
  : Add `public:md5`, `public:sha1`, and `public:sha256` secondary properties to record those hashes for the public key. Add `private:md5`, `private:sha1`, and `private:sha256` secondary properties to record those hashes for the public key.

  `geo:nloc`
  : The `geo:nloc` form has been deprecated.

  `geo:telem`
  : Add a new form to record a the location of a given node at a given time. This replaces the use of `geo:nloc`.

  `it:sec:c2:config`
  : Add a `proxies` secondary property to record proxy URLS used to communicate to a C2 server. Add a `listens` secondary property to record urls the software should bind. Add a `dns:resolvers` secondary property to record DNS servers the software should use. Add a `http:headers` secondary property to record HTTP headers the software should use.

  `it:exec:query`
  : Add a new form to record an instance of a query executed on a host.

  `it:query`
  : Add a new form to record query strings.

  `mat:type`
  : Add a taxonomy type to record taxonomies of material specifications or items.

  `mat:item`
  : Add a `type` secondary property to record the item type.

  `mat:spec`
  : Add a `type` secondary property to record the item type.

  `media:news`
  : Add a `publisher` secondary property to record the org that published the news. Add a `publisher:name` secondary property to record the name of the org. Deprecate the `org` secondary property.

  `ou:campaign`
  : Add a `conflict` secondary property to record the primary conflict associated the campaign.

  `ou:conflict`
  : Add a new form to record a conflict between two or more campaigns which have mutually exclusive goals.

  `ou:contribution`
  : Add a new form to represent contributing material support to a campaign.

  `pol:election`
  : Add a new form to record an election.

  `pol:race`
  : Add a new form to record indivdual races in an election.

  `pol:office`
  : Add a new form to record an appointed or elected office.

`pol:term`
: Add a new form to record the term in office for an individual.

`pol:candidate`
: Add a form to record a candidate for a given race.

`pol:pollingplace`
: Add a form to record the polling locations for a given election.

`proj:ticket`
: Add a `ext:creator` secondary form to record contact information from and external system.

- Annotate the following light edges. (#2757)

  `about`
  : A light edge created by the Storm `note.add` command, which records the relationship between a `meta:note` node and the target node.

  `includes`
  : When used with a `ou:contribution` node, the edge indicates the target node was the contribution made.

  `has`
  : When used with a `meta:ruleset` and `meta:rule` node, indicates the ruleset contains the rule.

  `matches`
  : When used with a `meta:rule` node, the edge indicates the target node matches the rule.

  `refs`
  : A light edge where the source node refers to the target node.

  `seenat`
  : When used with a `geo:telem` target node, the edge indicates the source node was seen a given location.

  `uses`
  : When used with a `ou:org` node, the edge indicates the target node is used by the organization.

- Commonly used light edges are now being annotated in the model, and are available through Cortex APIs which expose the data model. (#2757)
- Make Storm command argument parsing errors into exceptions. Previously the argument parsing would cause the Storm runtime to be torn down with `print` messages, which could be missed. This now means that automations which have a invalid Storm command invocation will fail loudly. (#2769)
- Allow a Storm API caller to set the task identifier by setting the `task` value in the Storm `opts` dictionary. (#2768) (#2774)
- Add support for registering and exporting custom STIX objects with the `$lib.stix` Storm APIS. (#2773)
- Add APIS and Storm APIs for enumerating mirrors that have been registered with AHA. (#2760)

17.75.2 Bugfixes

- Ensure that auto-adds are created when merging part of a View when using the Storm `merge --apply` command. (#2770)
- Add missing support for handling timezone offsets without colon separators when normalizing `time` values. `time` values which contain timezone offsets and not enough data to resolve minute level resolution will now fail to parse. (#2772)
- Fix an issue when normalizing `inet:url` values when the host value was the IPv4 address `0.0.0.0`. (#2771)
- Fix an issue with the Storm `cron.list` command, where the command failed to run when a user had been deleted. (#2776)

17.75.3 Improved Documentation

- Update the Storm user documentation to include the Embedded Property syntax, which is a shorthand (`::`) that can be used to reference properties on adjacent nodes. (#2767)
- Update the Synapse Glossary. (#2767)
- Update Devops documentation to clarify the Aha URLs which end with``...`` are intentional. (#2775)

17.76 v2.101.1 - 2022-07-14

17.76.1 Bugfixes

- Fix an issue where the Storm `scrape` command could fail to run with inbound nodes. (#2761)
- Fix broken links in documentation. (#2763)
- Fix an issue with the Axon `AxonHttpBySha256V1` API handler related to detecting `Range` support in the Axon. (#2764)

17.77 v2.101.0 - 2022-07-12

17.77.1 Automatic Migrations

- Create nodes in the Cortex for the updated properties noted in the data model updates listed below.
- Axon indices are migrated to account for storing offset information to support the new offset and size API options.
- See *Data Migration* for more information about automatic migrations.

17.77.2 Features and Enhancements

- Updates to the `crypto`, `infotech`, `ps`, and `transport` models. (#2720) (#2738) (#2739) (#2747)

  crypto:smart:effect:minttoken
  Add a new form to model smart contract effects which create non-fungible tokens.

  crypto:smart:effect:burntoken`
  Add a new form to model smart contract effects which destroy non-fungible tokens.

  crypto:smart:effect:proxytoken
  Add a new form that tracks grants for a non-owner address the ability to manipulate a specific non-fungible token.

  crypto:smart:effect:proxytokenall
  Add a new form that tracks grants for a non-owner address the ability to manipulate all of the non-fungible tokens.

  crypto:smart:effect:proxytokens
  Add a new form that tracks grants for a non-owner address to manipulate fungible tokens.

  it:av:signame
  Add a new form to track AV signature names. Migrate `it:av:filehit:sig:name` and `it:av:sig:name` to use the new form.

`it:exec:proc`
: Add a `name` secondary property to track the display name of a process. Add a `path:base` secondary property to track the basename of the executable for the process.

`ps:contact`
: Add an `orgnames` secondary property to track an array of orgnames associated with a contact.

`transport:sea:vessel`
: Add `make` and `model` secondary properties to track information about the vessel.

- Add a new Storm command, `movenodes`, that can be used to move a node entirely from one layer to another. (#2714)
- Add a new Storm library, `$lib.gen`, to assist with creating nodes based on secondary property based deconfliction. (#2754)
- Add a `sorted()` method to the `stat:tally` object, to simplify handling of tallied data. (#2748)
- Add a new Storm function, `$lib.mime.html.totext()`, to extract inner tag text from HTML strings. (#2744)
- Add Storm functions `$lib.crypto.hashes.md5()`, `$lib.crypto.hashes.sha1()`, `$lib.crypto.hashes.sha256()` and `$lib.crypto.hashes.sha512()` to allow hashing bytes directly in Storm. (#2743)
- Add an `Axon.csvrows()` API for streaming CSV rows from an Axon, and a corresponding `$lib.axon.csvrows()` Storm API. (#2719)
- Expand Synapse requirements to include updated versions of the `pycryptome`, `pygments`, and `scalecodec` modules. (#2752)
- Add range support to `Axon.get()` to read bytes from a given offset and size. The `/api/v1/axon/files/by/sha256/<SHA-256>` HTTP API has been updated to support a `Range` header that accepts a `bytes` value to read a subset of bytes that way as well. (#2731) (#2755) (#2758)

17.77.3 Bugfixes

- Fix `$lib.time.parse()` when `%z` is used in the format specifier. (#2749)
- Non-string form-data fields are now serialized as JSON when using the `Axon.postfiles()` API. (#2751) (#2759)
- Fix a byte-alignment issue in the `Axon.readlines()` API. (#2719)

17.78 v2.100.0 - 2022-06-30

17.78.1 Features and Enhancements

- Support parsing CVSS version 3.1 prefix values. (#2732)

17.78.2 Bugfixes

- Normalize tag value lists in `snap.addTag()` to properly handle JSON inputs from HTTP APIs. (#2734)
- Fix an issue that allowed multiple concurrent streaming backups to occur. (#2725)

17.78.3 Improved Documentation

- Add an entry to the devops task documentation for trimming Nexus logs. (#2730)
- Update the list of available Rapid Power-Ups. (#2735)

17.79 v2.99.0 - 2022-06-23

17.79.1 Features and Enhancements

- Add an extensible STIX 2.1 import library, `$lib.stix.import`. The function `$lib.stix.import.ingest()` can be used to STIX bundles into a Cortex via Storm. (#2727)
- Add a Storm `uptime` command to display the uptime of a Cortex or a Storm Service configured on the Cortex. (#2728)
- Add `--view` and `--optsfile` arguments to `synapse.tools.csvtool`. (#2726)

17.79.2 Bugfixes

- Fix an issue getting the maximum available memory for a host running with Linux cgroupsv2 apis. (#2728)

17.80 v2.98.0 - 2022-06-17

17.80.1 Features and Enhancements

- Updates to the `econ` model. (#2717)

  `econ:acct:balance`
  : Add `total:received` and `total:sent` properties to record total currency sent and received by the account.

- Add additional debug logging for Aha provisioning. (#2722)
- Adjust whitespace requirements on Storm grammar related to tags. (#2721)
- Always run the function provided to the Storm `divert` command per node. (#2718)

17.80.2 Bugfixes

- Fix an issue that prevented function arguments named `func` in Storm function calls. (#2715)
- Ensure that active coroutines have been cancelled when changing a Cell from active to passive status; before starting any passive coroutines. (#2713)
- Fix an issue where `Nexus._tellAhaReady` was registering with the Aha service when the Cell did not have a proper Aha service name set. (#2723)

17.81 v2.97.0 - 2022-06-06

17.81.1 Features and Enhancements

- Add an `/api/v1/aha/provision/service` HTTP API to the Aha service. This can be used to generate `aha:provision` URLs. (#2707)
- Add `proxy` options to `$lib.inet.http` Storm APIs, to allow an admin user to specify an alternative (or to disable) proxy setting. (#2706)
- Add a `--tag` and `--prop` option to the Storm `diff` command. Update the Storm `merge` command examples to show more real-world use cases. (#2710)
- Add the ability to set the layers in a non-forked view with the `$view.set(layers, $iden)` API on the Storm view object. (#2711)
- Improve Storm parser logic for handling list and expression syntax. (#2698) (#2708)

17.81.2 Bugfixes

- Improve error handling of double quoted strings in Storm when null characters are present in the raw query string. This situation now raises a BadSyntax error instead of an opaque Python ValueError. (#2709)
- Fix unquoted JSON keys which were incorrectly allowed in Storm JSON style expression syntax. (#2698)
- When merging layer data, add missing permission checks for light edge and node data changes. (#2671)

17.82 v2.96.0 - 2022-05-31

17.82.1 Features and Enhancements

- Updates to the `transport` model. (#2697)

  `velocity`
  : Add a new base type to record velocities in millimeters/second.

  `transport:direction`
  : Add a new type to indicate a direction of movement with respect to true North.

  `transport:air:telem`
  : Add `:course` and `:heading` properties to record the direction of travel. Add `:speed`, `:airspeed` and `:verticalspeed` properties to record the speed of travel.

`transport:sea:telem`
Add `:course` and `:heading` properties to record the direction of travel. Add a `:speed` property to record the speed of travel. Add `:destination`, `:destination:name` and `:destination:eta` to record information about the destination.

- Restore the precedence of environment variables over `cell.yaml` options during Cell startup. API driven overrides are now stored in the `cell.mods.yaml` file. (#2699)
- Add `--dmon-port` and `--https-port` options to the `synapse.tools.aha.provision.service` tool in order to specify fixed listening ports during provisioning. (#2703)
- Add the ability of `synapse.tools.moduser` to set user passwords. (#2695)
- Restore the call to the `recover()` method on the Nexus during Cell startup. (#2701)
- Add `mesg` arguments to `NoSuchLayer` exceptions. (#2696)
- Make the LMDB slab startup more resilient to a corrupted `cell.opts.yaml` file. (#2694)

17.82.2 Bugfixes

- Fix missing variable checks in Storm. (#2702)

17.82.3 Improved Documentation

- Add a warning to the deployment guide about using Docker on Mac OS. (#2700)

17.83 v2.95.1 - 2022-05-24

17.83.1 Bugfixes

- Fix a regression in the Telepath `aha://` update from `v2.95.0`. (#2693)

17.84 v2.95.0 - 2022-05-24

17.84.1 Features and Enhancements

- Add a `search` mode to Storm. The `search` mode utilizes the Storm search interface to lift nodes. The `lookup` mode no longer uses the search interface. (#2689)
- Add a `?mirror=true` flag to `aha://` Telepath URLs which will cause the Aha service lookups to prefer using a mirror of the service rather than the leader. (#2681)
- Add `$lib.inet.http.urlencode()` and `$lib.inet.http.urldecode()` Storm APIs for handling URL encoding. (#2688)
- Add type validation for all Cell configuration options throughout the lifetime of the Cell and all operations which modify its configuration values. This prevents invalid values from being persisted on disk. (#2687) (#2691)

17.84.2 Bugfixes

- Fix an issue where the = sign in the Storm grammar was assigned an anonymous terminal name by the grammar parser. This caused an issue with interpreting various syntax errors. (#2690)

17.85 v2.94.0 - 2022-05-18

17.85.1 Automatic Migrations

- Re-normalize the migrated properties noted in the data model updates listed below. See *Data Migration* for more information about automatic migrations.

17.85.2 Features and Enhancements

- Updates to the `crypto`, `infotech`, `ou`, and `person` models. (#2620) (#2684)

  `crypto:algorithm`
  Add a form to represent a named cryptography algorithm.

  `crypto:key`
  Add a form to represent a cryptographic key and algorithm.

  `crypto:smart:effect:transfertoken`
  Add a form to represent the effect of transferring ownership of a non-fungible token.

  `crypto:smart:effect:transfertokens`
  Add a form to represent the effect of transferring multiple fungible tokens.

  `crypto:smart:effect:edittokensupply`
  Add a form to represent the increase or decrease in the supply of fungible tokens.

  `it:prod:softname`
  Add a form to represent a software name.

  `it:host`
  Add a `:os:name` secondary property.

  `it:mitre:attack:software`
  Migrate the `:name` and `:names` properties to `it:prod:softname` type.

  `it:prod:soft`
  Migrate the `:name` and `:names` properties to `it:prod:softname` type.

  `it:prod:softver`
  Deprecate the `:software:name` property. Migrate the `:name` and `:names` properties to `it:prod:softname` type.

  `it:app:yara:rule`
  Add a `:family` property to represent the software family the rule is designed to detect.

  `it:sec:c2:config`
  Add a form to represent C2 configuration data.

  `ou:campaign`
  Add a `:org:name` property to represent the name of the organization responsible the campaign. Add a `:org:fqdn` property to represent the fqdn of the organization responsible the campaign. Add a `:team` property to represent the team responsible for the campaign.

`ou:team`
Add a form to represent a team within an organization.

`ou:industry`
Migrate the `:name` property to `ou:industryname` type. Add a `:names` property for alternative names.

`ou:industryname`
Add a form to represent the name of an industry.

`ou:position`
Add a `:team` property to represent the team associated with a given position.

`ps:contact`
Add a `:crypto:address` property to represent the crypto currency address associated with the contact.

- Add `$lib.copy()` to Storm. This allows making copies of objects which are compatible with being serialized with msgpack. (#2678)
- Remove *print* events from the Storm *limit* command. (#2674)

17.85.3 Bugfixes

- Fix an issue where client certificates presented in Telepath `ssl` connections could fallback to resolving users by a prefix. This was not intended to be allowed when client certificates are used with Telepath. (#2675)
- Fix an issue where `node:del` triggers could fail to fire when adding nodeedits directly to a view or snap. (#2654)
- Fix header escaping when generating autodoc content for Synapse Cells. (#2677)
- Assorted unit tests fixes to make tests more stable. (#2680)
- Fix an issue with Storm function argument parsing. (#2685)

17.85.4 Improved Documentation

- Add an introduction to Storm libraries and types. (#2670) (#2683)
- Fix small typos and corrections in the devops documentation. (#2673)

17.86 v2.93.0 - 2022-05-04

17.86.1 Features and Enhancements

- Updates to the `inet` and `infotech` models. (#2666)

`:sandbox:file`
Add a `sandbox:file` property to record an initial sample from a sandbox environment to the following forms:

`it:exec:proc` `it:exec:thread` `it:exec:loadlib` `it:exec:mmap` `it:exec:mutex` `it:exec:pipe` `it:exec:url` `it:exec:bind` `it:exec:file:add` `it:exec:file:del` `it:exec:file:read` `it:exec:file:write` `it:exec:reg:del` `it:exec:reg:get` `it:exec:reg:set`

`it:host:activity`
Update the interface to add a `sandbox:file` property to record an initial sample from a sandbox environment.

- Changed primary Storm parser to a LALR compatible syntax to gain 80x speed up in parsing Storm queries (#2649)
- Added service provisioning API to AHA service and associated tool `synapse.tools.aha.provision.service` and documentation to make it easy to bootstrap Synapse services using service discovery and SSL client-side certificates to identify service accounts. (#2641)
- Added user provisioning API to AHA service and associated tools `synapse.tools.aha.provision.user` and `synapse.tools.aha.enroll` to make it easy to bootstrap new users with SSL client-side certificates and AHA service discovery configuration. (#2641)
- Added automatic mirror initialization logic to Synapse services to enable new mirrors to be initilized dynamically via AHA provisioning rather than from a pre-existing backup. (#2641)
- Added `handoff()` API to Synapse services to allow mirrors to be gracefully promoted to leader. (#2641)
- Added `synapse.tools.promote` to allow easy promotion of mirror to leader using the new `handoff()` API. (#2641)
- Added `aha:provision` configuration to Synapse services to allow them to automatically provision and self-configure using AHA. (#2641)
- Adjusted Synapse service configuration preference to allow runtime settings to be stored in `cell.yaml`. (#2641)
- Added optional `certhash` parameter to telepath `ssl://` URLs to allow cert-pinning behavior and automatic trust of provisioning URLs. (#2641)
- Added `synapse.tools.moduser` and `synapse.tools.modrole` commands to modernize and ease user/role management from within Synapse service docker containers. (#2641)
- Add `$lib.jsonstor.cacheget()` and `lib.jsonstor.cacheset()` functions in Storm to easily implement data caching in the JSONStor. (#2662)
- Add a `params` option to `$lib.inet.http.connect()` to pass parameters when creating Websocket connections in Storm. (#2664)

17.86.2 Bugfixes

- Added `getCellRunId()` API to Synapse services to allow them to detect incorrect mirror configurations where they refer to themselves. (#2641)
- Ensure that CLI history files can be read and written upon starting interactive CLI tools. (#2660)
- Assorted unit tests fixes to make tests more stable. (#2656) (#2665)
- Fix several uses of Python features which are formally deprecated and may be removed in future Python versions. (#2668)

17.86.3 Improved Documentation

- Added new Deployment Guide with step-by-step production ready deployment instructions (#2641)
- Refactored Devops Guide to give task-oriented instructions on performing common devops tasks. (#2641)
- Added new minimal Admin Guide as a place for documenting Cortex admin tasks. (#2641)
- Updated Getting Started to direct users to synapse-quickstart instructions. (#2641)
- Added `easycert` tool documentation. (#2641)
- Removed `cmdr` tool documentation to emphasize newer tools such as `storm`. (#2641)

- Update the list of available Advanced and Rapid Power-Ups. (#2667)

17.87 v2.92.0 - 2022-04-28

17.87.1 Features and Enhancements

- Update the allowed versions of the `pyopenssl` and `pytz` libraries. (#2657) (#2658)

17.87.2 Bugfixes

- When setting ival properties, they are now properly merged with existing values. This only affected multi-layer views. (#2655)

17.88 v2.91.1 - 2022-04-24

17.88.1 Bugfixes

- Fix a parsing regression in inet:url nodes related to unencoded "@" symbols in URLs. (#2653)

17.89 v2.91.0 - 2022-04-21

17.89.1 Features and Enhancements

- Updates to the `inet` and `infotech` models. (#2634) (#2644) (#2652)

  `inet:url`
  The `inet:url` type now recognizes various `file:///` values from RFC 8089.

  `it:sec:cve`
  The `it:sec:cve` type now replaces various Unicode dashes with hyphen characters when norming. This allows a wider range of inputs to be accepted for the type. Scrape related APIs have also been updated to match on this wider range of inputs.

- The Cell now uses `./backup` as a default path for storing backups in, if the `backup:dir` path is not set. (#2648)
- Add POSIX advisory locking around the Cell `cell.guid` file, to prevent multiple processes from attempting to start a Cell from the same directory. (#2642)
- Change the default `SLAB_COMMIT_WARN` time from 5 seconds to 1 second, in order to quickly identify slow storage performance. (#2630)
- Change the Cell `iterBackupArchive` and `iterNewBackupArchive` routines to always log exceptions they encounter, and report the final log message at the appropriate log level for success and failure. (#2629)
- When normalizing the `str` types, when `onespace` is specified, we skip the `strip` behavior since it is redundant. (#2635)
- Log exceptions raised by Cell creation in `initFromArgv`. Catch `lmdb.LockError` when opening a LMDB database and re-raise an exception with a clear error message. (#2638)
- Update schema validation for Storm packages to ensure that cmd arguments do not have excess fields in them. (#2650)

17.89.2 Bugfixes

- Adjust comma requirements for the JSON style list and dictionary expressions in Storm. (#2636)
- Add Storm query logging in a code execution path where it was missing. (#2647)
- Tuplify the output of `synapse.tools.genpkg.loadPkgProto` to ensure that Python list constructs `[...]` do not make it into Power-Up documentation. (#2646)
- Fix an issue with heavy Stormtypes objects where caching was preventing some objects from behaving in a dynamic fashion as they were intended to. (#2640)
- In norming `int` values, when something is outside of the minimum or maximum size of the type, we now include the string representation of the valu instead of the raw value. (#2643)
- Raise a `NotReady` exception when a client attempts to resolve an `aha://` URL and there have not been any `aha` servers registered. (#2645)

17.89.3 Improved Documentation

- Update Storm command reference to add additional commands. (#2633)
- Expand Stormtypes API documentation. (#2637) (#2639)

17.90 v2.90.0 - 2022-04-04

17.90.1 Features and Enhancements

- Updates to the `meta` and `infotech` models. (#2624)

  `meta:rule`
  Add a new form for generic rules, which should be linked to the nodes they match with a `matches` light edge.

  `meta:ruleset`
  Add `:author`, `:created`, and `:updated` secondary properties.

  `it:app:yara:rule`
  Add `:created` and `:updated` secondary properties.

- Add a new Docker image `vertexproject/synapse-jsonstor`. (#2627)
- Allow passing a version requirement string to `$lib.import()`. (#2626)

17.90.2 Bugfixes

- Fix an issue where using a regex lift on an array property could incorrectly yield the same node multiple times. (#2625)

17.90.3 Improved Documentation

- Update documentation regarding mirroring to be clearer about whether a given cell supports it. (#2619)

17.91 v2.89.0 - 2022-03-31

17.91.1 Features and Enhancements

- Update the `meta` model. (#2621)

  `meta:ruleset`
  Add a new form to denote the collection of a set of nodes representing rules, which should be linked together with a `has` light edge.

- Add additional filter options for the Storm `merge` command. (#2615)
- Update the `BadSyntaxError` exception thrown when parsing Storm queries to additionally include line and column when available. Fix an issue where a `!` character being present in the exception text could truncate the output. (#2618)

17.92 v2.88.0 - 2022-03-23

17.92.1 Automatic Migrations

- Re-normalize the `geo:place:name`, `crypto:currency:block:hash`, and `crypto:currency:transaction:hash` values to account for their modeling changes. Migrate `crypto:currency:transaction:input` and `crypto:currency:transaction:output` values to the secondary properties on the respective `crypto:payment:input` and `crypto:payment:output` nodes to account for the modeling changes. Make `geo:name` nodes for `geo:place:name` secondary properties to account for the modeling changes. See *Data Migration* for more information about automatic migrations.

17.92.2 Features and Enhancements

- Several updates for the `crypto`, `geospace`, `inet`, and `meta` models. (#2594) (#2608) (#2611) (#2616)

  `crypto:payment:input`
  Add a secondary property `:transaction` to denote the transaction for the payment.

  `crypto:payment:output`
  Add a secondary property `:transaction` to denote the transaction for the payment.

  `crypto:currency:block`
  Change the type of the `:hash` property from a `0x` prefixed `str` to a `hex` type.

  `crypto:currency:transaction`
  Change the type of the `:hash` property from a `0x` prefixed `str` to a `hex` type. Deprecate the `:inputs` and `:outputs` secondary properties.

  `geo:place`
  Change the type of the `:name` secondary property to `geo:name`.

  `inet:web:channel`
  Add a new form to denote a channel within a web service or instance.

inet:web:instance
Add a new form to track an instance of a web service, such as a channel based messaging platform.

inet:web:mesg
Add `:channel`, `:place`, and `:place:name` secondary properties.

inet:web:post
Add `:channel` and `:place:name` secondary properties.

meta:event
Add a new form to denote an analytically relevant event in a curated timeline.

meta:event:taxonomy
Add a new form to represent a taxonomy of `meta:event:type` values.

meta:timeline
Add a new form to denote a curated timeline of analytically relevant events.

meta:timeline:taxonomy
Add a new form to represent a taxonomy of `meta:timeline:type` values.

- Add support for `$lib.len()` to count the length of emitter or generator functions. (#2603)
- Add support for scrape APIs to handle text that has been defanged with `\\.` characters. (#2605)
- Add a `nomerge` option to View objects that can be set to prevent merging a long lived fork. (#2614)
- Add `liftByProp()` and `liftByTag()` methods to the Stormtypes `layer` objects. These allow lifting of nodes based on data stored in a specific layer. (#2613)
- Expand Synapse requirements to include updated versions of the `pygments` library. (#2602)

17.92.3 Improved Documentation

- Fix the example regular expressions used in the `$lib.scrape.genMatches()` Storm library API examples. (#2606)

17.93 v2.87.0 - 2022-03-18

17.93.1 Features and Enhancements

- Several updates for the `inet` and `meta` models. (#2589) (#2592)

  inet:ssl:jarmhash
  Add a form to record JARM hashes.

  inet:ssl:jarmsample
  Add a form to record JARM hashes being present on a server.

  meta:note
  Add a form for recording free text notes.

- Update the Synapse docker containers to be built from a Ubuntu based image, instead of a Debian based image. (#2596)
- Add a Storm `note.add` command that creates a `meta:note` node to record freeform text, and links that node to the input nodes using a `about` light edge. (#2592)
- Support non-writeable or non-existing directories within Synapse `certdir` directories. (#2590)

- Add an optional `tick` argument to the `synapse.lib.lmdbslab.Hist.add()` function. This is exposed internally for Axon implementations to use. (#2593)
- Expand Synapse requirements to include updated versions of the `pycryptome`, `pygments`, `scalecodec` and `xxhash` modules. (#2598)

17.93.2 Bugfixes

- Fix an issue where the StormDmon stop/start status was not properly being updated in the runtime object, despite being properly updated in the Hive. (#2598)
- Calls to `addUnivProp()` APIs when the universal property name already exists now raise a `DupPropName` exception. (#2601)

17.94 v2.86.0 - 2022-03-09

17.94.1 Automatic Migrations

- Migrate secondary properties in Cortex nodes which use `hugenum` type to account for updated ranges. See *Data Migration* for more information about automatic migrations.

17.94.2 Features and Enhancements

- Extend the number of decimal places the `hugenum` type can store to 24 places, with a new maximum value of 730750818665451459101842. (#2584) (#2586)
- Update `fastjsonschema` to version `2.15.3`. (#2581)

17.94.3 Bugfixes

- Add missing read-only flags to secondary properties of Comp type forms which were computed from the primary property of the node. This includes the following: (#2587)
  - `crypto:currency:address:coin`
  - `crypto:currency:address:iden`
  - `crypto:currency:block:coin`
  - `crypto:currency:block:offset`
  - `crypto:currency:client:coinaddr`
  - `crypto:currency:client:inetaddr`
  - `crypto:currency:smart:token:contract`
  - `crypto:currency:smart:token:tokenid`
  - `crypto:x509:revoked:crl`
  - `crypto:x509:revoked:cert`
  - `crypto:x509:signedfile:cert`
  - `crypto:x509:signedfile:file`

  - `econ:acquired:item`
  - `econ:acquired:purchase`
  - `inet:dns:query:client`
  - `inet:dns:query:name`
  - `inet:dns:query:type`
  - `inet:whois:contact:type`
  - `inet:wifi:ap:bssid`
  - `inet:wifi:ap:ssid`
  - `mat:itemimage:file`
  - `mat:itemimage:item`
  - `mat:specimage:file`
  - `mat:specimage:spec`
  - `ou:id:number:type`
  - `ou:id:number:value`
  - `ou:hasgoal:goal`
  - `ou:hasgoal:org`
  - `tel:mob:cell:carrier`
  - `tel:mob:cell:carrier:mcc`
  - `tel:mob:cell:carrier:mnc`
  - `tel:mob:cell:cid`
  - `tel:mob:cell:lac`
- Fix an issue where Layers configured with writeback mirrors did not properly handle results which did not have any changes. (#2583)

17.94.4 Improved Documentation

- Fix spelling issues in documentation and API docstrings. (#2582) (#2585)

17.95 v2.85.1 - 2022-03-03

17.95.1 Bugfixes

- Fix a permission enforcement issue in autoadd mode that allowed users with view read permissions to add automatically detected and validated nodes but make no further edits. (#2579)
- Log errors encountered in the Layer mirror loop which don't have a local caller waiting on the change. (#2580)

17.96 v2.85.0 - 2022-03-03

17.96.1 Features and Enhancements

- Several updates for the `crypto`, `geo`, `inet`, `it`, `ps` and `risk` models. (#2570) (#2573) (#2574)

  `crypto:payment:input`
  : Add a new form to record payments made into a transaction.

  `crypto:payment:output`
  : Add a new form to record payments receieved from a transaction.

  `crypto:currency:transaction`
  : Add `inputs` and `outputs` array secondary properties to record inputs and outputs for a given transaction.

  `geo:name`
  : Add a new form representing an unstructured place name or address.

  `geo:place`
  : Add a `names` secondary property which is an array of `geo:name` values.

  `inet:flow`
  : Add `dst:txcount`, `src:txcount`, `tot:txcount` and `tot:txbytes` secondary properties.

  `it:exec:proc`
  : Add an `account` secondary property as a `it:account` type. Mark the `user` secondary property as deprecated.

  `ps:contact`
  : Add `birth:place`, `birth:place:loc`, `birth:place:name`, `death:place`, `death:place:loc` and `death:place:name` secondary properties.

  `risk:compromise`
  : Add a `theft:price` secondary property to represent value of stolen assets.

- Embed Cron, StormDmon, and Trigger iden values and automation types into the Storm runtime when those automations are run. This information is populated in a dictionary variable named `$auto`. (#2565)
- Add `$lib.crypto.coin.ethereum.eip55()` to convert an Ethereum address to a checksummed address. (#2577)
- Add a `default` argument to the `$lib.user.allowed()` and `allowed()` method on `user` StormType. (#2570)
- Add a `inaugural` configuration key to the base `Cell` class. This can currently be used to bootstrap roles, permissions, and users in a Cell upon the first time it is started. (#2570)
- De-duplicate nodes when running the Storm `lookup` mode to lift nodes. (#2567)
- Add a test helper that can be used to isolate the `synapse.lib.certdir.certdir` singleton behavior via context manager. (#2564)

17.96.2 Bugfixes

- Calls to `addFormProp()` APIs when the property name already exists now raise a `DupPropName` exception. (#2566)
- Do not allow Storm `macro`'s to be created that have names greater than 492 characters in length. (#2569)
- Fix a bug in the scrape logic for Ethereum where the regular expression matched on `0X` prefixed strings but the validation logic did not account for that uppercase character. (#2575)

17.96.3 Improved Documentation

- Add documentation for the `$auto` variable embedded into the Cron, StormDmon, and Trigger automations. Add documentation for variables representing the form, node value, properties and tags which are responsible for Triggers running. (#2565)

17.97 v2.84.0 - 2022-02-22

17.97.1 Features and Enhancements

- Add `$lib.time.toUTC()` to adjust a local epoch milliseconds time to UTC. (#2550)
- Add a optional `timeout` argument to `$lib.service.wait()`. The function now returns `$lib.true` if the service is available, or `$lib.false` if the service does not become available during the timeout window. (#2561)
- Update the `Layer.verify()` routines to add verification of tagprop and array indexes in layers. These routines are in a beta status and are subject to change. (#2560)
- Update the Cortex's connection to a remote Axon to use a Telepath Client. (#2559)

17.98 v2.83.0 - 2022-02-17

17.98.1 Features and Enhancements

- Add `:ip:proto` and `:ip:tcp:flags` properties to the `inet:flow` form. (#2554)
- Add `$lib.log.debug()`, `$lib.log.info()`, `$lib.log.warning()`, and `$lib.log.error()` Stormtypes APIs. These allow a user to send log messages to the Cortex logging output directly.
- Update the `synapse.tools.genpkg` tool to support using files with the `.storm` extension. This is enabled by adding the following option to a Storm package definition. (#2555)

```
genopts:
  dotstorm: true
```

- Add form and prop values to `BadTypeValu` exceptions when raised during node edit generation. (#2552)

17.98.2 Bugfixes

- Correct a race condition in the `CoreApi.syncLayersEvents` and `CoreApi.syncIndexEvents` APIs. (#2553)

17.98.3 Improved Documentation

- Remove outdated documentation related to making `CoreModule` classes. (#2556)

17.99 v2.82.1 - 2022-02-11

17.99.1 Bugfixes

- Re-order node edit validation to only check read-only status of properties if the value would change. (#2547)
- Raise the correct exception when parsing invalid time values, like `0000-00-00`. (#2548)
- Disable node caching for `StormDmon` runtimes to avoid potential cache coherency issues. (#2549)

17.100 v2.82.0 - 2022-02-10

17.100.1 Features and Enhancements

- Add an `addNode()` API to the Stormtypes `view` object. This allows the programmatic creation of a node with properties being set in a transactional fashion. (#2540)
- Add support to Storm for creating JSON style list and dictionary objects. (#2544)
- The `AhaCell` now bootstraps TLS CA certificates for the configured `aha:network` value, a host certificate for the `aha:name` value, and a user certificate for the `aha:admin` value. (#2542)
- Add `mesg` arguments to all exceptions raised in `synapse.lib.certdir`. (#2546)

17.100.2 Improved Documentation

- Fix some missing and incorrect docstrings for Stormtypes. (#2545)

17.100.3 Deprecations

- Telepath APIs and Storm commands related to `splices` have been marked as deprecated. (#2541)

17.101 v2.81.0 - 2022-01-31

17.101.1 Features and Enhancements

- The `it:sec:cpe` now recognizes CPE 2.2 strings during type normalization. CPE 2.2 strings will be upcast to CPE 2.3 and the 2.2 string will be added to the `:v2_2` secondary property of `it:sec:cpe`. The Storm hotfix `$lib.cell.hotFixesApply()` can be used to populate the `:v2_2` property on existing `it:sec:cpe` nodes where it is not set. (#2537) (#2538) (#2539)
- Setting properties on nodes may now take a fast path if the normed property has no subs, no autoadds and is not a locked property. (#2539)

17.101.2 Bugfixes

- Fix an issue with `Ival` `norm()` routines when norming a tuple or list of values. The max value returned previously could have exceeded the value of the future marker `?`, which would have been then caused an a `BadTypeValu` exception during node edit construction. This is is now caught during the initial `norm()` call. (#2539)

17.102 v2.80.1 - 2022-01-26

17.102.1 Bugfixes

- The embedded JsonStor added to the Cortex in `v2.80.0` needed to have a stable iden for the Cell and and auth subsystem. This has been added. (#2536)

17.103 v2.80.0 - 2022-01-25

17.103.1 Features and Enhancements

- Add a triple quoted string `'''` syntax to Storm for defining multiline strings. (#2530)
- Add a JSONStor to the Cortex, and expose that in Storm for storing user related content. (#2530) (#2513)
- Add durable user notifications to Storm that can be used to send and receive messages between users. (#2513)
- Add a `leaf` argument to `$node.tags()` that causes the function to only return the leaf tags. (#2535)
- Add an error message in the default help text in pure Storm commands when a user provides additional arguments or switches, in addition to the `--help` switch. (#2533)
- Update `synapse.tools.genpkg` to automatically bundle Optic workflows from files on disk. (#2531)
- Expand Synapse requirements to include updated versions of the `packaging`, `pycryptome` and `scalecodec` modules. (#2534)

17.103.2 Bugfixes

- Add a missing `tostr()` call to the Storm `background` query argument. (#2532)

17.104 v2.79.0 - 2022-01-18

17.104.1 Features and Enhancements

- Add `$lib.scrape.ndefs()` and `$lib.scrape.context()` to scrape text. The `ndefs()` API yields a unique set of node form and value pairs, while the `context()` API yields node form, value, and context information for all matches in the text. (#2508)
- Add `:name` and `:desc` properties to the `it:prod:softver` form. (#2528)
- Update the `Layer.verify()` routines to reduce false errors related to array types. The method now takes a dictionary of configuration options. These routines are in a beta status and are subject to change. (#2527)
- Allow setting a View's parent if does not have an existing parent View and only has a single layer. (#2515)
- Add `hxxp[:\\]` and `hxxps[:\\]` to the list of known defanging strategies which are identified and replaced during text scraping. (#2526)
- Expand Synapse requirements to include updated versions of the `typing-extensions` module. (#2525)

17.104.2 Bugfixes

- Storm module interfaces now populate `modconf` data when loaded. (#2508)
- Fix a missing keyword argument from the `AxonApi.wput()` method. (#2527)

17.104.3 Deprecations

- The `$lib.scrape()` function has been deprecated in favor the new `$lib.scrape` library functions. (#2508)

17.105 v2.78.0 - 2022-01-14

17.105.1 Automatic Migrations

- Migrate Cortex nodes which may have been skipped in an earlier migration due to missing tagprop indexes. See *Data Migration* for more information about automatic migrations.

17.105.2 Features and Enhancements

- Expand Synapse requirements to include updated versions of the `base58`, `cbor2`, `lmdb`, `pycryptodome`, `PyYAML`, `xxhash`. (#2520)

17.105.3 Bugfixes

- Fix an issue with the Tagprop migration from `v2.42.0` where a missing index could have resulted in Layer storage nodes not being updated. (#2522) (#2523)
- Fix an issue with `synapse.lib.platforms.linux.getTotalMemory()` when using a process segregated with the Linux cgroups2 API. (#2517)

17.105.4 Improved Documentation

- Add devops instructions related to automatic data migrations for Synapse components. (#2523)
- Update the model deprecation documentation for the `it:host:model` and `it:host:make` properties. (#2521)

17.106 v2.77.0 - 2022-01-07

17.106.1 Features and Enhancements

- Add Mach-O metadata support the file model. This includes the following new forms: `file:mime:macho:loadcmd`, `file:mime:macho:version`, `file:mime:macho:uuid`, `file:mime:macho:segment`, and `file:mime:macho:section`. (#2503)
- Add `it:screenshot`, `it:prod:hardware`, `it:prod:component`, `it:prod:hardwaretype`, and `risk:mitigation` forms to the model. Add `:hardware` property to `risk:hasvuln` form. Add `:hardware` property to `it:host` form. The `:manu` and `:model` secondary properties on `it:host` have been deprecated. (#2514)
- The `guid` type now strips hyphen (`-`) characters when doing norm. This allows users to provide external UUID / GUID strings for use. (#2514)
- Add a `Axon.postfiles()` to allow POSTing files as multi-part form encoded files over HTTP. This is also exposed through the `fields` argument on the Storm `$lib.inet.http.post()` and `$lib.inet:http:request` APIs. (#2516)
- Add `.yu` ccTLD to the list of TLDs identified by the Synapse scrape functionality. (#2518)
- Add `mesg` arguments to all instances of `NoSuchProp` exceptions. (#2519)

17.107 v2.76.0 - 2022-01-04

17.107.1 Features and Enhancements

- Add `emit` and `stop` keywords to Storm. The `emit` keyword is used in functions to make them behave as generators, which can yield arbitrary values. The `stop` keyword can be used to prematurely end a function which is `emit`'ing values. (#2475)

- Add Storm Module Interfaces. This allows Storm Package authors to define common module interfaces, so that multiple modules can implement the API convention to provide a consistent set of data across multiple Storm modules. A `search` convention is added to the Cortex, which will be used in `lookup` mode when the `storm:interface:search` configuration option is set. (#2475)
- Storm queries in `lookup` mode now fire `look:miss` events into the Storm message stream when the lookup value contains a valid node value, but the node is not present in the current View. (#2475)
- Add a `:host` secondary property to `risk:hasvuln` form to record `it:host` instances which have a vulnerability. (#2512)
- Add `synapse.lib.scrape` support for identifying `it:sec:cve` values. (#2509)

17.107.2 Bugfixes

- Fix an `IndexError` that can occur during `Layer.verify()` routines. These routines are in a beta status and are subject to change. (#2507)
- Ensure that parameter and header arguments passed to Storm `$lib.inet.http` functions are cast into strings values. (#2510)

17.108 v2.75.0 - 2021-12-16

This release contains an automatic data migration that may cause additional startup time on the first boot. This is done to unique array properties which previously were not uniqued. Deployments with startup or liveliness probes should have those disabled while this upgrade is performed to prevent accidental termination of the Cortex process. Please ensure you have a tested backup available before applying this update.

17.108.1 Features and Enhancements

- Update the following array properties to be unique sets, and add a data model migration to update the data at rest: (#2469)
  - `biz:rfp:requirements`
  - `crypto:x509:cert:ext:sans`
  - `crypto:x509:cert:ext:crls`
  - `crypto:x509:cert:identities:fqdns`
  - `crypto:x509:cert:identities:emails`
  - `crypto:x509:cert:identities:ipv4s`
  - `crypto:x509:cert:identities:ipv6s`
  - `crypto:x509:cert:identities:urls`
  - `crypto:x509:cert:crl:urls`
  - `inet:whois:iprec:contacts`
  - `inet:whois:iprec:links`
  - `inet:whois:ipcontact:roles`
  - `inet:whois:ipcontact:links`
  - `inet:whois:ipcontact:contacts`

  - `it:account:groups`
  - `it:group:groups`
  - `it:reveng:function:impcalls`
  - `it:reveng:filefunc:funccalls`
  - `it:sec:cve:references`
  - `risk:vuln:cwes`
  - `tel:txtmesg:recipients`

- Add Layer index verification routines, to compare the Layer indices against the stored data for Nodes. This is exposed via the `.verify()` API on the Stormtypes `layer` object. These routines are in a beta status and are subject to change. (#2488)
- The `.json()` API on `inet:http:resp` now raises a `s_exc.BadJsonText` exception, which can be caught with the Storm `try ... catch` syntax. (#2500)
- Add `$lib.inet.ipv6.expand()` to expand an IPv6 address to its long form. (#2502)
- Add `hasPathObj()`, `copyPathObj()` and `copyPathObjs()` APIs to the `JsonStor`. (#2438)
- Allow setting a custom title when making documentation for Cell `confdefs` with the `synapse.tools.autodoc` tool. (#2504)
- Update the minimum version of the `aiohttp` library to `v3.8.1`. (#2495)

17.108.2 Improved Documentation

- Add content previously hosted at `commercial.docs.vertex.link` to the mainline Synapse documentation. This includes some devops information related to orchestration, information about Advanced and Rapid Power-Ups, information about the Synapse User Interface, as well as some support information. (#2498) (#2499) (#2501)
- Add `Synapse-Malshare` and `Synapse-TeamCymru` Rapid Power-Ups to the list of available Rapid Power-Ups. (#2506)
- Document the `jsonlines` option for the `api/v1/storm` and `api/v1/storm/nodes` HTTP APIs. (#2505)

17.109 v2.74.0 - 2021-12-08

17.109.1 Features and Enhancements

- Add `.onion` and `.bit` to the TLD list used for scraping text. Update the TLD list from the latest IANA TLD list. (#2483) (#2497)
- Add support for writeback mirroring of layers. (#2463) (#2489)
- Add `$lib.scrape()` Stormtypes API. This can be used to do programmatic scraping of text using the same regular expressions used by the Storm `scrape` command and the `synapse.lib.scrape` APIs. (#2486)
- Add a `jsonlines` output mode to Cortex streaming HTTP endpoints. (#2493)
- Add a `--raw` argument to the Storm `pkg.load` command. This loads the raw JSON response as a Storm package. (#2491)
- Add a `blocked` enum to the `proj:ticket:status` property to represent a blocked ticket. (#2490)

17.109.2 Bugfixes

- Fix a behavior with `$path` losing variables in pure Storm command execution. (#2492)

17.109.3 Improved Documentation

- Update the description of the Storm `scrape` command. (#2494)

17.110 v2.73.0 - 2021-12-02

17.110.1 Features and Enhancements

- Add a Storm `runas` command. This allows admin users to execute Storm commands as other users. (#2473)
- Add a Storm `intersect` command. This command produces the intersection of nodes emitted by running a Storm query over all inbound nodes to the `intersect` command. (#2480)
- Add `wait` and `timeout` parameters to the `Axon.hashes()` and `$lib.axon.list()` APIs. (#2481)
- Add a `readonly` flag to `synapse.tools.genpkg.loadPkgProto()` and `synapse.tools.genpkg.tryLoadPkgProto()` APIs. If set to `True` this will open files in read only mode. (#2485)
- Allow Storm Prim objects to be capable of directly yielding nodes when used in `yield` statements. (#2479)
- Update the StormDmon subsystem to add debug log information about state changes, as well as additional data for structured logging output. (#2455)

17.110.2 Bugfixes

- Catch a fatal application error that can occur in the Cortex if the forked process pool becomes unusable. Previously this would cause the Cortex to appear unresponsive for executing Storm queries; now this causes the Cortex to shut down gracefully. (#2472)
- Fix a Storm path variable scoping issue where variables were improperly scoped when nodes were passed into pure Storm commands. (#2459)

17.111 v2.72.0 - 2021-11-23

17.111.1 Features and Enhancements

- Update the cron subsystem logs to include the cron name, as well as adding additional data for structured logging output. (#2477)
- Add a `sort_keys` argument to the `$lib.yaml.save()` Stormtype API. (#2474)

17.111.2 Bugfixes

- Update the `asyncio-socks` version to a version which has a pinned version range for the `python-socks` dependency. (#2478)

17.112 v2.71.1 - 2021-11-22

17.112.1 Bugfixes

- Update the `PyOpenSSL` version to `21.0.0` and pin a range of modern versions of the `cryptography` which have stronger API compatibility. This resolves an API compatibility issue with the two libraries which affected SSL certificate generation. (#2476)

17.113 v2.71.0 - 2021-11-19

17.113.1 Features and Enhancements

- Add support for asynchronous triggers. This mode of trigger operation queues up the trigger event in the View for eventual processing. (#2464)
- Update the crypto model to add a `crypto:smart:token` form to represent a token managed by a smart contract. (#2462)
- Add `$lib.axon.readlines()` and `$lib.axon.jsonlines()` to Stormtypes. (#2468)
- Add the Storm `mode` to the structured log output of a Cortex executing a Storm query. (#2466)

17.113.2 Bugfixes

- Fix an error when converting Lark exceptions to Synapse `BadSyntaxError`. (#2471)

17.113.3 Improved Documentation

- Revise the Synapse documentation layout. (#2460)
- Update type specific behavior documentation for `time` types, including the recently added wildcard time syntax. (#2467)
- Sort the Storm Type documentation by name. (#2465)
- Add 404 handler pages to our documentation. (#2461) (#2470)

17.113.4 Deprecations

- Remove `$path.trace()` objects. (#2445)

17.114 v2.70.1 - 2021-11-08

17.114.1 Bugfixes

- Fix an issue where `$path.meta` data was not being properly serialized when heavy Stormtype objects were set on the `$path.meta` dictionary. (#2456)
- Fix an issue with Stormtypes `Str.encode()` and `Bytes.decode()` methods when handling potentially malformed Unicode string data. (#2457)

17.114.2 Improved Documentation

- Update the Storm Control Flow documentation with additional examples. (#2443)

17.115 v2.70.0 - 2021-11-03

17.115.1 Features and Enhancements

- Add `:dst:handshake` and `src:handshake` properties to `inet:flow` to record text representations of the handshake strings of a given connection. (#2451)
- Add a `proj:attachment` form to the `project` model to represent attachments to a given `proj:ticket`. (#2451)
- Add a implicit wildcard behavior to the `time` type when lifting or filtering nodes. Dates ending in a `*` are converted into ranges covering all possible times in them. For example, `.created=202101*` would lift all nodes created on the first month of 2021. (#2446)
- Add the following `$lib.time` functions to chop information from a time value. (#2446)
  - `$lib.time.year()`
  - `$lib.time.month()`
  - `$lib.time.day()`
  - `$lib.time.hour()`
  - `$lib.time.minute()`
  - `$lib.time.second()`
  - `$lib.time.dayofweek()`
  - `$lib.time.dayofmonth()`
  - `$lib.time.monthofyear()`
- Add `List.extend()`, `List.slice()`, `Str.find()`, and `Str.size()` functions to Stormtypes. (#2450) (#2451)
- Add `$lib.json.schema()` and a `json:schema` object to Stormtypes. These can be used to validate arbitrary data JSON structures in Storm using JSON Schema. (#2448)

- Update syntax checking rules and address deprecation warnings for strings in the Synapse codebase. (#2426)

17.116 v2.69.0 - 2021-11-02

17.116.1 Features and Enhancements

- Add support for building Optic Workflows for Storm Packages in the `synapse.tools.genpkg` tool. (#2444)
- The `synapse.tools.storm` CLI tool now prints out node properties in precedence order. (#2449)
- Update the global Stormtypes registry to better track types when they are added or removed. (#2447)

17.117 v2.68.0 - 2021-10-29

17.117.1 Features and Enhancements

- Add `crypto:currency:transaction`, `crypto:currency:block`, `crypto:smart:contract` and `econ:acct:balanc` forms. (#2423)
- Add `$lib.hex.decode()` and `$lib.hex.encode()` Stormtypes functions to encode and decode hexidecimal data as bytes. Add `slice()` and `unpack()` methods to the Storm Bytes object. (#2441)
- Add `$lib.yaml` and `$lib.xml` Stormtypes libraries for interacting with YAML and XML text, respectively. (#2434)
- Add a Storm `version` command to show the user the current version of Synapse the Cortex is using. (#2440)

17.117.2 Bugfixes

- Fix overzealous `if` statement caching in Storm. (#2442)

17.118 v2.67.0 - 2021-10-27

17.118.1 Features and Enhancements

- Add `$node.addEdge()` and `$node.delEdge()` APIs in Storm to allow for programatically setting edges. Add a `reverse` argument to `$node.edges()` that allows traversing edges in reverse. (#2351)

17.118.2 Bugfixes

- Fix a pair of regressions related to unicode/IDNA support for scraping and normalizing FQDNs. (#2436)

17.118.3 Improved Documentation

- Add documentation for the Cortex `api/v1/storm/call` HTTP API endpoint. (#2435)

17.119 v2.66.0 - 2021-10-26

17.119.1 Features and Enhancements

- Improve unicode/IDNA support for scraping and normalizing FQDNs. (#2408)
- Add `$lib.inet.http.ouath` to support OAuth based workflows in Storm, starting with OAuth v1.0 support. (#2413)
- Replace `pysha3` requirement with `pycryptodome`. (#2422)
- Add a `tls:ca:dir` configuration option to the Cortex and Axon. This can be used to provide a directory of CA certificate files which are used in Storm HTTP API and Axon wget/wput APIs. (#2429)

17.119.2 Bugfixes

- Catch and raise bad ctors given in RStorm `storm-cortex` directives. (#2424)
- Fix an issue with the `cron.at` command not properly capturing the current view when making the Cron job. (#2425)
- Disallow the creation of extended properties, universal properties, and tag properties which are not valid properties in the Storm grammar. (#2428)
- Fix an issue with `$lib.guid()` missing a `toprim()` call on its input. (#2421)

17.119.3 Improved Documentation

- Update our Cell devops documentation to note how to replace the TLS keypair used by the built in webserver with third party certificates. (#2432)

17.120 v2.65.0 - 2021-10-16

17.120.1 Features and Enhancements

- Add support for interacting with IMAP email servers though Storm, using the `$lib.inet.imap.connect()` function. This returns a object that can be used to delete, read, and search emails in a given IMAP mailbox. (#2399)
- Add a new Storm command, `once`. This command can be used to 'gate' a node in a Storm pipeline such that the node only passes through the command exactly one time for a given named 'gate'. The gate information is stored in nodedata, so it is inspectable and subject to all other features that apply to nodedata. (#2404)
- Add a `:released` property to `it:prod:softver` to record when a software version was released. (#2419)
- Add a `tryLoadPkgProto` convenience function to the `synapse.tools.genpkg` for Storm service package generation with inline documentation. (#2414)

17.120.2 Bugfixes

- Add `asyncio.sleep(0)` calls in the `movetag` implementation to address some possible hot-loops. (#2411)
- Clarify and sanitize URLS in a Aha related log message i `synapse.telepath`. (#2415)

17.120.3 Improved Documentation

- Update our `fork` definition documentation. (#2409)
- Add documentation for using client-side TLS certificates in Telepath. (#2412)
- Update the Storm CLI tool documentation. (#2406)
- The Storm types and Storm library documentation now automatically links from return values to return types. (#2410)

17.121 v2.64.1 - 2021-10-08

17.121.1 Bugfixes

- Add a retry loop in the base `Cell` class when attempting to register with an `Aha` server. (#2405)
- Change the behavior of `synapse.common.yamlload()` to not create files when the expected file is not present on disk, and open existing files in read-only mode. (#2396)

17.122 v2.64.0 - 2021-10-06

17.122.1 Features and Enhancements

- Add support for scraping the following cryptocurrency addresses to the `synapse.lib.scrape` APIs and Storm `scrape` command. (#2387) (#2401)
  - Bitcoin
  - Bitcoin Cash
  - Ethereum
  - Ripple
  - Cardano
  - Polkadot

  The internal cache of regular expressions in the `synapse.lib.scrape` library is also now a private member; API users should use the `synapse.lib.scrape.scrape()` function moving forward.
- Add `:names` property to the `it:mitre:attack:software` form. (#2397)
- Add a `:desc` property to the `inet:whois:iprec` form. (#2392)
- Added several new Rstorm directives. (#2359) (#2400)
  - `storm-cli` - Runs a Storm query with the Storm CLI tool
  - `storm-fail` - Toggles whether or not the following Storm command should fail or not.

- storm-multiline - Allows embedding a multiline Storm query as a JSON encoded string for future execution.
- storm-vcr-callback - Allows specifying a custom callback which a VCR object is sent too.

17.122.2 Bugfixes

- Fix a missing toprim() call when loading a Storm package directly with Storm. (#2359)
- Fix a caching issue where tagprops were not always being populated in a Node tagprop dictionary. (#2396)
- Add a mesg argument to a few NoSuchVar and BadTypeValu exceptions. (#2403)

17.122.3 Improved Documentation

- Storm reference docs have been converted from Jupyter notebook format to Synapse .rstorm format, and now display examples using the Storm CLI tool, instead of the Cmdr CLI tool. (#2359)

17.123 v2.63.0 - 2021-09-29

17.123.1 Features and Enhancements

- Add a risk:attacktype taxonomy to the risk model. Add :desc and :type properties to the risk:attack form. (#2386)
- Add :path property to the it:prod:softfile form. (#2388)

17.123.2 Bugfixes

- Fix the repr for the``auth:user`` Stormtype when printing a user object in Storm. (#2383)

17.124 v2.62.1 - 2021-09-22

17.124.1 Bugfixes

- Fix an issue in the Nexus log V1 to V2 migration code which resulted in LMDB file copies being made instead of having directories renamed. This can result in a sparse file copy of the Nexus log, resulting in a condition where the volume containing the Cell directory may run out of space. (#2374)

17.125 v2.62.0 - 2021-09-21

17.125.1 Features and Enhancements

- Add APIs to support trimming, rotating and culling Nexus logs from Cells with Nexus logging enabled. These operations are distributed to downstream consumers, of the Nexus log (e.g. mirrors). For the Cortex, this can be invoked in Storm with the `$lib.cell.trimNexsLog()` Stormtypes API. The Cortex devops documentation contains more information about Nexus log rotation. (#2339) (#2371)
- Add `.size()` API to the Stormtypes `storm:query` object. This will run the query and return the number of nodes it would have yielded. (#2363)

17.125.2 Improved Documentation

- Document the tag glob meanings on the Stormtypes `$node.tags()` API. (#2368)

17.126 v2.61.0 - 2021-09-17

17.126.1 Features and Enhancements

- Add a `!export` command to the Storm CLI to save query results to a `.nodes` file. (#2356)
- Add `$lib.cell.hotFixesCheck()` and `$lib.cell.hotFixesApply()` Stormtypes functions. These can be used to apply optional hotfixes to a Cortex on demand by an admin. (#2348)
- Add `$lib.infosec.cvss.calculateFromProps()` to allow calculating a CVSS score from a dictionary of CVSS properties. (#2353)
- Add `$node.data.has()` API to Stormtypes to allow easy checking if a node has nodedata for a given name. (#2350)

17.126.2 Bugfixes

- Fix for large return values with `synapse.lib.coro.spawn()`. (#2355)
- Fix `synapse.lib.scrape.scrape()` capturing various common characters used to enclose URLs. (#2352)
- Ensure that generators being yielded from are always being closed. (#2358)
- Fix docstring for `str.upper()` in Stormtypes. (#2354)

17.126.3 Improved Documentation

- Add link to the Power-Ups blog post from the Cortex dev-ops documentation. (#2357)

17.127 v2.60.0 - 2021-09-07

17.127.1 Features and Enhancements

- Add new `risk:compromise` and `risk:compromisetype` forms. Add `attacker`, `compromise`, and `target` secondary properties to the `risk:attack` form. (#2348)

17.127.2 Bugfixes

- Add a missing `wait()` call when calling the `CoreApi.getAxonUpload()` and `CoreApi.getAxonBytes()` Telepath APIs. (#2349)

17.127.3 Deprecations

- Deprecate the `actor:org`, `actor:person`, `target:org` and `target:person` properties on `risk:attack` in favor of new `attacker` and `target` secondary properties. Deprecate the `type` property on `ou:campaign` in favor of the `camptype` property. (#2348)

17.128 v2.59.0 - 2021-09-02

17.128.1 Features and Enhancements

- Add a new Storm command, `pkg.docs`, to enumerate any documentation that has been bundled with a Storm package. (#2341)
- Add support for manipulating `'proj:comment` nodes via Stormtypes. (#2345)
- Add `Axon.wput()` and `$lib.axon.wput()` to allow POSTing a file from an Axon to a given URL. (#2347)
- Add `$lib.export.toaxon()` to allow exporting a `.nodes` file directly to an Axon based on a given storm query and opts. (#2347)
- The `synapse.tools.feed` tool now accepts a `--view` argument to feed data to a specific View. (#2342)
- The `synapse.tools.feed` tool now treats `.nodes` files as msgpack files for feeding data to a Cortex. (#2343)
- When the Storm `help` command has an argument without any matching commands, it now prints a helpful message. (#2338)

17.128.2 Bugfixes

- Fix a caching issue between `$lib.lift.byNodeData()` and altering the existing node data on a given node. (#2344)
- Fix an issue with backups were known lmdbslabs could be omitted from being treated as lmdb databases, resulting in inefficient file copies being made. (#2346)

17.129 v2.58.0 - 2021-08-26

17.129.1 Features and Enhancements

- Add `!pushfile`, `!pullfile`, and `!runfile` commands to the `synapse.tools.storm` tool. (#2334)
- Add multiname SNI support to `ssl://` listening configurations for the Daemon. (#2336)
- Add a new Cortex HTTP API Endpoint, `/api/v1/feed`. This can be used to add nodes to the Cortex in bulk. (#2337)
- Refactor the `syn.nodes` feed API implementation to smooth out the ingest rate. (#2337)
- Sort the Storm Package commands in documentation created by `synpse.tools.autodoc` alphabetically. (#2335)

17.129.2 Deprecations

- Deprecate the `syn.splices` and `syn.nodedata` feed API formats. (#2337)

17.130 v2.57.0 - 2021-08-24

17.130.1 Features and Enhancements

- Add a basic `synapse.tools.storm` CLI tool. This can be used to connect to a Cortex via Telepath and directly execute Storm commands. (#2332)
- Add an `inet:http:session` form to track the concept of a prolonged session a user may have with a webserver across multiple HTTP requests. Add an `:success` property to the ``ou:campaign` form to track if a campaign was sucessful or not. Add an `:goal` property to the `risk:attack` form to track the specific goal of the attack. Add an `:desc` property to the `proj:project` form to capture a description of the project. (#2333)

17.130.2 Bugfixes

- Fix an issue with `synapse.lib.rstorm` where multiline node properties could produce RST which did not render properly. (#2331)

17.130.3 Improved Documentation

- Clean up the documentation for the Storm `wget` command. (#2325)

17.131 v2.56.0 - 2021-08-19

17.131.1 Features and Enhancements

- Refactor some internal Axon APIs for downstream use. (#2330)

17.131.2 Bugfixes

- Resolve an ambiguity in the Storm grammar with yield statement and dollar expressions inside filter expression. There is a slight backwards incompatibility with this change, as dollar expressions insider of filter expressions now require a `$` prepended where before it was optional. (#2322)

17.132 v2.55.0 - 2021-08-18

17.132.1 Features and Enhancements

- Add `$node.props.set()` Stormtypes API to allow programmatically setting node properties. (#2324)
- Deny non-runtsafe invocations of the following Storm commands: (#2326)
  - `graph`
  - `iden`
  - `movetag`
  - `parallel`
  - `tee`
  - `tree`
- Add a `Axon.hashset()` API to get the md5, sha1, sha256 and sha512 hashes of file in the Axon. This is exposed in Stormtypes via the `$lib.bytes.hashset()` API. (#2327)
- Add the `synapse.servers.stemcell` server and a new Docker image, `vertexproject/synaspe-stemcell`. The Stemcell server is similar to the `synapse.servers.cell` server, except it resolves the Cell ctor from the `cell:ctor` key from the `cell.yaml` file, or from the `SYN_STEM_CELL_CTOR` environment variable. (#2328)

17.133 v2.54.0 - 2021-08-05

17.133.1 Features and Enhancements

- Add `storm-envvar` directive to RST preprocessor to include environment variables in `storm-pre` directive execution context. (#2321)
- Add new `diff` storm command to allow users to easily lift the set of nodes with changes in the top layer of a forked view. Also adds the `--no-tags` option to the `merge` command to allow users to omit `tag:add` node edits and newly constructed `syn:tag` nodes when merging selected nodes. (#2320)
- Adds the following properties to the data model: (#2319)
  - `biz:deal:buyer:org`

  - biz:deal:buyer:orgname
  - biz:deal:buyer:orgfqdn
  - biz:deal:seller:org
  - biz:deal:seller:orgname
  - biz:deal:seller:orgfqdn
  - biz:prod:madeby:org
  - biz:prod:madeby:orgname
  - biz:prod:madeby:orgfqdn
  - ou:opening:posted
  - ou:opening:removed
  - ou:org:vitals
- Updates `storm-mock-http` to support multiple HTTP requests/responses in RST preprocessor. (#2317)

17.134 v2.53.0 - 2021-08-05

This release contains an automatic data migration that may cause additional startup time on the first boot. This is done to unique array properties which previously were not uniqued. Deployments with startup or liveliness probes should have those disabled while this upgrade is performed to prevent accidental termination of the Cortex process. Please ensure you have a tested backup available before applying this update.

17.134.1 Features and Enhancements

- Add an `embeds` option to Storm to allow extracting additional data when performing queries. (#2314)
- Enforce node data permissions at the Layer boundary. Remove the `node.data.get` and `node.data.list` permissions. (#2311)
- Add `auth.self.set.email`, `auth.self.set.name`, `auth.self.set.passwd` permissions on users when changing those values. These permissions default to being allowed, allowing a rule to be created that can deny users from changing these values. (#2311)
- Add `$lib.inet.smtp` to allow sending email messages from Storm. (#2315)
- Warn if a LMDB commit operation takes too long. (#2316)
- Add new data types, `taxon` and `taxonomy`, to describe hierarchical taxonomies. (#2312)
- Add a new Business Development model. This allows tracking items related to contract, sales, and purchasing lifecycles. This adds the following new forms to the data model: `biz:dealtype`, `biz:prodtype`, `biz:dealstatus`, `biz:rfp`, `biz:deal`, `biz:bundle`, `biz:product`, and `biz:stake`. The Org model is also updated to add new forms for supporting parts of the business lifecycle, adding `ou:jobtype`, `ou:jobtitle`, `ou:employment`, `ou:opening`, `ou:vitals`, `ou:camptype`, and `ou:orgtype`, `ou:conttype` forms. The Person model got a new form, `ps:workhist`. (#2312)
- Add a `:deleted` property to `inet:web:post`. (#2312)
- Update the following array properties to be unique sets, and add a data model migration to update the data at rest: (#2312)
  - edu:course:prereqs

  - `edu:class:assistants`
  - `ou:org:subs`
  - `ou:org:names`
  - `ou:org:dns:mx`
  - `ou:org:locations`
  - `ou:org:industries`
  - `ou:industry:sic`
  - `ou:industry:subs`
  - `ou:industry:isic`
  - `ou:industry:naics`
  - `ou:preso:sponsors`
  - `ou:preso:presenters`
  - `ou:conference:sponsors`
  - `ou:conference:event:sponsors`
  - `ou:conference:attendee:roles`
  - `ou:conference:event:attendee:roles`
  - `ou:contract:types`
  - `ou:contract:parties`
  - `ou:contract:requirements`
  - `ou:position:reports`
  - `ps:person:names`
  - `ps:person:nicks`
  - `ps:persona:names`
  - `ps:persona:nicks`
  - `ps:education:classes`
  - `ps:contactlist:contacts`

17.134.2 Bugfixes

- Prevent renaming the `all` role. (#2313)

17.134.3 Improved Documentation

- Add documentation about Linux kernel parameteres which can be tuned to affect Cortex performance. (#2316)

17.135 v2.52.1 - 2021-07-30

17.135.1 Bugfixes

- Fix a display regression when enumerating Cron jobs with the Storm `cron.list` command. (#2309)

17.136 v2.52.0 - 2021-07-29

17.136.1 Features and Enhancements

- Add a new specification for defining input forms that a pure Storm command knows how to natively handle. (#2301)
- Add `Lib.reverse()` and `Lib.sort()` methods to Stormtypes API. (#2306)
- Add `View.parent` property in Stormtypes API. (#2306)
- Support Telepath Share objects in Storm. (#2293)
- Allow users to specify a view to run a cron job against, move a cron job to a new view, and update permission check for adding/moving cron jobs to views. (#2292)
- Add CPE and software name infomation to the `inet:flow` form. Add `it:av:prochit`, `it:exec:thread`, `it:exec:loadlib`, `it:exec:mmap`, `it:app:yara:procmatch` forms to the infotech model. Add `:names` arrays to `it:prod:soft` and `it:prod:softver` forms to assist in entity resolution of software. Add a `risk:alert` form to the risk model to allow for capturing arbitrary alerts. (#2304)
- Allow Storm packages to specify other packages they require and possible conflicts would prevent them from being installed in a Cortex. (#2307)

17.136.2 Bugfixes

- Specify the View when lifting `syn:trigger` runt nodes. (#2300)
- Update the scrape URL regular expression to ignore trailing periods and commas. (#2302)
- Fix a bug in Path scope for nodes yielding by pure Storm commands. (#2305)

17.137 v2.51.0 - 2021-07-26

17.137.1 Features and Enhancements

- Add a `--size` option to the Storm `divert` command to limit the number of times the generator is iterated. (#2297)
- Add a `perms` key to the pure Storm command definition. This allows for adding intuitive permission boundaries for pure Storm commands which are checked prior to command execution. (#2297)

- Allow full properties with comparators when specifying the destination or source when walking light edges. (#2298)

17.137.2 Bugfixes

- Fix an issue with LMDB slabs not being backed up if their directories did not end in `.lmdb`. (#2296)

17.138 v2.50.0 - 2021-07-22

17.138.1 Features and Enhancements

- Add `.cacheget()` and `cacheset()` APIs to the Storm `node:data` object for easy caching of structured data on nodes based on time. (#2290)
- Make the Stormtypes unique properly with a Set type. This does disallow the use of mutable types such as dictionaries inside of a Set. (#2225)
- Skip executing non-runtsafe commands when there are no inbound nodes. (#2291)
- Add `asroot:perms` key to Storm Package modules. This allows package authors to easily declare permissions their packages. Add Storm commands `auth.user.add`, `auth.role.add`, `auth.user.addrule`, `auth.role.addrule`, and `pkg.perms.list` to help with some of the permission management. (#2294)

17.139 v2.49.0 - 2021-07-19

17.139.1 Features and Enhancements

- Add a `iden` parameter when creating Cron jobs to allow the creation of jobs with stable identifiers. (#2264)
- Add `$lib.cell` Stormtypes library to allow for introspection of the Cortex from Storm for Admin users. (#2285)
- Change the Telepath Client connection loop error logging to log at the Error level instead of the Info level. (#2283)
- Make the tag part normalization more resilient to data containing non-word characters. (#2289)
- Add `$lib.tags.prefix()` Stormtypes to assist with normalizing a list of tags with a common prefix. (#2289)
- Do not allow the Storm `divert` command to work with non-generator functions. (#2282)

17.139.2 Bugfixes

- Fix an issue with Storm command execution with non-runtsafe options. (#2284)
- Log when the process pool fails to initialize. This may occur in certain where CPython multiprocessing primitives are not completely supported. (#2288)
- In the Telepath Client, fix a race condition which could have raised an AttributeError in Aha resolutions. (#2286)
- Prevent the reuse of a Telepath Client object when it has been fini'd. (#2286)
- Fix a race condition in the Aha server when handling distributed changes which could have left the service in a desynchronized state. (#2287)

17.139.3 Improved Documentation

- Update the documentation for the `synapse.tools.feed` tool. (#2279)

17.140 v2.48.0 - 2021-07-13

17.140.1 Features and Enhancements

- Add a Storm `divert` command to ease the implementation of `--yield` constructs in Storm commands. This optionally yields nodes from a generator, or yields inbound nodes, while still ensuring the generator is conusmed. (#2277)
- Add Storm runtime debug tracking. This is a boolean flag that can be set or unset via `$lib.debug`. It can be used by Storm packages to determine if they should take extra actions, such as additional print statements, without needing to track additional function arguments in their implementations. (#2278)

17.140.2 Bugfixes

- Fix an ambiguity in the Storm grammar. (#2280)
- Fix an issue where form autoadds could fail to be created in specific cases of the model. (#2273)

17.141 v2.47.0 - 2021-07-07

17.141.1 Features and Enhancements

- Add `$lib.regex.replace()` Stormtypes API to perform regex based replacement of string parts. (#2274)
- Add universal properties to the dictionary returned by `Cortex.getModelDict()` as a `univs` key. (#2276)
- Add additional `asyncio.sleep(0)` statements to `Layer._storNodeEdits` to improve Cortex responsiveness when storing large numbers of edits at once. (#2275)

17.142 v2.46.0 - 2021-07-02

17.142.1 Features and Enhancements

- Update the Cortex `storm:log:level` configuration value to accept string values such as `DEBUG`, `INFO`, etc. The default log level for Storm query logs is now `INFO` level. (#2262)
- Add `$lib.regex.findall()` Stormtypes API to find all matching parts of a regular expression in a given string. (#2265)
- Add `$lib.inet.http.head()` Stormtypes API to perform easy HEAD requests, and `allow_redirects` arguments to existing `lib.inet.http` APIs to allow controlling the redirect behavior. (#2268)
- Add `$lib.storm.eval()` API to evaluate Storm values from strings. (#2269)
- Add `getSystemInfo()` and `getBackupInfo()` APIS to the Cell for getting useful system information. (#2267)
- Allow lists in rstorm bodies. (#2261)

- Add a `:desc` secondary property to the `proj:sprint` form. (#2261)
- Call _normStormPkg in all loadStormPkg paths, move validation to post normalization and remove mutation in validator (#2260)
- Add `SYN_SLAB_COMMIT_PERIOD` environment variable to control the Synapse slab commit period. Add `layer:lmdb:max_replay_log` Cortex option to control the slab replay log size. (#2266)
- Update Ahacell log messages. (#2270)

17.142.2 Bugfixes

- Fix an issue where the `Trigger.pack()` method failed when the user that created the trigger had been deleted. (#2263)

17.142.3 Improved Documentation

- Update the Cortex devops documentation for the Cortex to document the Storm query logging. Update the Cell devops documentation to explain the Cell logging and how to enable structured (JSON) logging output. (#2262)
- Update Stormtypes API documentation for `bool`, `proj:epic`, `proj:epics`, `proj:ticket`, `proj:tickets`, `proj:sprint`, `proj:sprints`, `proj:project`, `stix:bundle` types. (#2261)

17.143 v2.45.0 - 2021-06-25

17.143.1 Features and Enhancements

- Add a application level process pool the base Cell implemenation. Move the processing of Storm query text into the process pool. (#2250) (#2259)
- Minimize the re-validation of Storm code on Cortex boot. (#2257)
- Add the `ou:preso` form to record conferences and presentations. Add a `status` secondary property to the `it:mitre:attack:technique` form to track if techniques are current, deprecated or withdrawn. (#2254)

17.143.2 Bugfixes

- Remove incorrect use of `cmdopts` in Storm command definitions unit tests. (#2258

17.144 v2.44.0 - 2021-06-23

This release contains an automatic data migration that may cause additional startup time on the first boot. This only applies to a Cortex that is using user defined tag properties or using `ps:person:name` properties. Deployments with startup or liveliness probes should have those disabled while this upgrade is performed to prevent accidental termination of the Cortex process. Please ensure you have a tested backup available before applying this update.

17.144.1 Features and Enhancements

- Add a `.move()` method on Stormtypes `trigger` objects to allow moving a Trigger from one View to another View. (#2252)
- When the Aha service marks a service as down, log why that service is being marked as such. (#2255)
- Add `:budget:price` property to the `ou:contract` form. Add `:settled` property to the `econ:purchase` form. (#2253

17.144.2 Bugfixes

- Make the array property `ps:person:names` a unique array property. (#2253
- Add missing tagprop key migration for the bybuidv3 index. (#2256)

17.145 v2.43.0 - 2021-06-21

17.145.1 Features and Enhancements

- Add a `.type` string to the Stormtypes `auth:gate` object to allow a user to identify the type of auth gate it is. (#2238)
- Add `$lib.user.iden` reference to the Stormtype `$lib.user` to get the iden of the current user executing Storm code. (#2236)
- Add a `--no-build` option to `synapse.tools.genpkg` to allow pushing an a complete Storm Package file. (#2231) (#2232) (#2233)
- The Storm `movetag` command now checks for cycles when setting the `syn:tag:isnow` property. (#2229)
- Deprecate the `ou:org:has` form, in favor of using light edges for storing those relationships. (#2234)
- Add a `description` property to the `ou:industry` form. (#2239)
- Add a `--name` parameter to the Storm `trigger.add` command to name triggers upon creation. (#2237)
- Add `regx` to the `BadTypeValu` exception of the `str` type when a regular expression fails to match. (#2240)
- Consolidate Storm parsers to a single Parser object to improve startup time. (#2247)
- Improve error logging in the Cortex `callStorm()` and `storm()` APIs. (#2243)
- Add `from:contract`, `to:contract`, and `memo` properties to the `econ:acct:payment` form. (#2248)
- Improve the Cell backup streaming APIs link cleanup. (#2249)

17.145.2 Bugfixes

- Fix issue with grabbing the incorrect Telepath link when performing a Cell backup. (#2246)
- Fix missing `toprim` calls in `$lib.inet.http.connect()`. (#2235)
- Fix missing Storm command form hint schema from the Storm Package schema. (#2242)

17.145.3 Improved Documentation

- Add documentation for deprecated model forms and properties, along with modeling alternatives. (#2234)
- Update documentation for the Storm `help` command to add examples of command substring matching. (#2241)

17.146 v2.42.2 - 2021-06-11

17.146.1 Bugfixes

- Protect against a few possible RuntimeErrors due to dictionary sizes changing during iteration. (#2227)
- Fix StormType `Lib` lookups with imported modules which were raising a `TypeError` instead of a `NoSuchName` error. (#2228)
- Drop old Storm Packages if they are present when re-adding them. This fixes an issue with runtime updates leaving old commands in the Cortex. (#2230)

17.147 v2.42.1 - 2021-06-09

17.147.1 Features and Enhancements

- Add a `--no-docs` option to the `synapse.tools.genpkg` tool. When used, this not embed inline documentation into the generated Storm packages. (#2226)

17.148 v2.42.0 - 2021-06-03

17.148.1 Features and Enhancements

- Add a `--headers` and `--parameters` arguments to the Storm `wget` command. The default headers now includes a browser like UA string. (#2208)
- Add the ability to modify the name of a role via Storm. (#2222)

17.148.2 Bugfixes

- Fix an issue in the JsonStor cell where there were missing fini calls. (#2223)
- Add a missing timeout to an `getAhaSvc()` call. (#2224)
- Change how tagprops are serialized to avoid a issue with sending packed nodes over HTTP APIs. This changes the packed node structure of tagprops from a dictionary keyed with `(tagname, propertyname)` to a dictionary keyed off of the `tagname`, which now points to a dictionary containing the `propertyname` which represents the value of the tagprop. (*#2221* <https://github.com/vertexproject/synapse/pull/2221>`_)

17.149 v2.41.1 - 2021-05-27

17.149.1 Bugfixes

- Add PR #2117 to bugfix list in CHANGLOG.rst for v2.41.0 :D

17.150 v2.41.0 - 2021-05-27

17.150.1 Features and Enhancements

- Add an `it:cmd` form and update the `it:exec:proc:cmd` property to use it. This release includes an automatic data migration on startup to update the `it:exec:proc:cmd` on any existing `it:exec:proc` nodes. (#2219)

17.150.2 Bugfixes

- Fix an issue where passing a Base object to a sub-runtime in Storm did not correctly increase the reference count. (#2216)
- Fix an issue where the `tee` command could potentially run the specified queries twice. (#2218)
- Fix for rstorm using mock when the HTTP body is bytes. (#2217)

17.151 v2.40.0 - 2021-05-26

17.151.1 Features and Enhancements

- Add a `--parallel` switch to the `tee` Storm command. This allows for all of the Storm queries provided to the `tee` command to execute in parallel, potentially producing a mixed output stream of nodes. (#2209)
- Convert the Storm Runtime object in a Base object, allowing for reference counted Storm variables which are made from Base objects and are properly torn down. (#2203)
- Add `$lib.inet.http.connect()` method which creates a Websocket object inside of Storm, allowing a user to send and receive messages over a websocket. (#2203)
- Support pivot join operations on tags. (#2213)
- Add `stormrepr()` implementation for `synapse.lib.stormtypes.Lib`, which allows for `$lib.print()` to display useful strings for Storm Libraries and imported modules. (#2212)
- Add a storm API top updated a user name. (#2214)

17.151.2 Bugfixes

- Fix the logger name for `synapse.lib.aha`. (#2210)
- Log `ImportError` exceptions in `synapse.lib.dyndeps.getDynMod`. This allows easier debugging when using the `synapse.servers.cell` server when running custom Cell implementations. (#2211)
- Fix an issue where a Storm command which failed to set command arguments successfully would not teardown the Storm runtime. (#2212)

17.152 v2.39.1 - 2021-05-21

17.152.1 Bugfixes

- Fix an issue with referencing the Telepath user session object prior to a valid user being set. (#2207)

17.153 v2.39.0 - 2021-05-20

17.153.1 Features and Enhancements

- Add more useful output to Storm when printing heavy objects with `$lib.print()`. (#2185)
- Check rule edits for roles against provided authgates in Storm. (#2199)
- Add `Str.rsplit()` and maxsplit arguments to `split()/rsplit()` APIs in Storm. (#2200)
- Add default argument values to the output of Storm command help output. (#2198)
- Add a `syn:tag:part` Type and allow the `syn:tag` type to normalize a list of tag parts to create a tag string. This is intended to be used with the `$lib.cast()` function in Storm. (#2192)
- Add debug logging to the Axon for reading, writing, or deleting of blobs. (#2202)
- Add a timeout argument to the `$lib.inet.http` functions. The functions will all now always return a `inet:http:resp` object; if the `.code` is -1, an unrecoverable exception occurred while making the request. (#2205)
- Add support for embedding a logo and documentation into a Storm Package. (#2204)

17.153.2 Bugfixes

- Fix export filters to correctly filter tagprops. (#2196)
- Fix an issue with Hotcount which prevented it from storing negative values. (#2197)
- Fix an issue where `hideconf` configuration values were being included in autodoc output. (#2199)

17.154 v2.38.0 - 2021-05-14

17.154.1 Features and Enhancements

- Remove trigger inheritance from Views. Views will now only execute triggers which are created inside of them. (#2189)
- Remove read-only property flags from secondary properties on `file:bytes` nodes. (#2191)
- Add a simple `it:log:event` form to capture log events. (#2195)
- Add structured logging as an option for Synapse Cells. When enabled, this produces logs as JSONL sent to stderr. This can be set via the `SYN_LOG_STRUCT` environment variable, or adding the `--structured-logging` command line switch. (#2179)
- Add a `nodes.import` command to import a `.nodes` file from a URL. (#2186)
- Allow the `desc` key to View and Layer objects in Storm. This can be used to set descriptions for these objects. (#2190)
- Use the gateiden in Storm auth when modifying rules; allowing users to share Views and Layers with other users. (#2194)

17.154.2 Bugfixes

- Fix an issue with Storm Dmon deletion not behaving properly in mirror configurations. (#2188)
- Explicitly close generators in Telepath where an exception has caused the generator to exit early. (#2183)
- Fix an issue where a trigger owner not having access to a view would cause the Storm pipeline to stop. (#2189)

17.155 v2.37.0 - 2021-05-12

17.155.1 Features and Enhancements

- Add a `file:mime:image` interface to the Synapse model for recording MIME specific metadata from image files. (#2187)
- Add `file:mime:jpg`, `file:mime:tiff`, `file:mime:gif` and `file:mime:png` specific forms for recording metadata of those file types. (#2187)
- Add `$lib.pkg.has()` Stormtype API to check for for the existence of a given Storm package by name. (#2182)
- All `None / $lib.null` as input to setting a user password. This clears the password and prevents a user from being able to login. (#2181)
- Grab any Layer push/pull offset values when calling `Layer.pack()`. (#2184)
- Move the retrieval of `https:headers` from HTTP API handlers into a function so that downstream implementers can redirect where the extra values are retrieved from. (#2187)

17.155.2 Bugfixes

- Fix an issue which allowed for deleted Storm Packages to be retrieved from memory. (#2182)

17.156 v2.36.0 - 2021-05-06

17.156.1 Features and Enhancements

- Add `risk:vuln` support to the default Stix 2.1 export, and capture vulnerability information used by threat actors and in campaigns. Add the ability to validate Stix 2.1 bundles to ensure that they are Stix 2.1 CS02 compliant. Add the ability to lift Synapse nodes based on bundles which were previously exported from Synapse. The lift feature only works with bundles created with Synapse v2.36.0 or greater. (#2174)
- Add a `Str.upper()` function for uppercasing strings in Storm. (#2174)
- Automatically bump a user's StormDmon's when they are locked or unlocked. (#2177)
- Add Storm Package support to `synapse.tools.autodocs` and update the rstorm implementation to capture additional directives. (#2172)
- Tighten lark-parser version requirements. (#2175)

17.156.2 Bugfixes

- Fix reported layer size to represent actual disk usage. (#2173)

17.157 v2.35.0 - 2021-04-27

17.157.1 Features and Enhancements

- Add `:issuer:cert` and `:selfsigned` properties to the `crypto:x509:cert` form to enable modeling X509 certificate chains. (#2163)
- Add a `https:headers` configuration option to the Cell to allow setting arbitrary HTTP headers for the Cell HTTP API server. (#2164)
- Update the Cell HTTP API server to have a minimum TLS version of v1.2. Add a default `/robots.txt` route. Add `X-XSS=Protection` and `X-Content-Type-Options` headers to the default HTTP API responses. (#2164)
- Update the minimum version of LMDB to `1.2.1`. (#2169)

17.157.2 Bugfixes

- Improve the error message for Storm syntax error handling. (#2162)
- Update the layer byarray index migration to account for arrays of `inet:fqdn` values. (#2165) (#2166)
- Update the `vertexproject/synapse-aha`, `vertexproject/synapse-axon`, `vertexproject/synapse-cortex`, and `vertexproject/synapse-cryotank` Docker images to use `tini` as a default entrypoint. This fixes an issue where signals were not properly being propagated to the Cells. (#2168)
- Fix an issue with enfanged indicators which were not properly being lifted by Storm when operating in `lookup` mode. (#2170)

17.158 v2.34.0 - 2021-04-20

17.158.1 Features and Enhancements

- Storm function definitions now allow keyword arguments which may have default values. These must be read-only values. (#2155) (#2157)
- Add a `getCellInfo()` API to the `Cell` and `CellAPI` classes. This returns metadata about the cell, its version, and the currently installed Synapse version. Cell implementers who wish to expose Cell specific version information must adhere to conventiosn documented in the API docstrings of the function. (#2151)
- Allow external Storm modules to be added in genpkg definitions. (#2159)

17.158.2 Bugfixes

- The `$lib.layer.get()` Stormtypes returned the top layer of the default view in the Cortex when called with no arguments, instead of the top layer of the current view. This now returns the top layer of the current view. (#2156)
- Avoid calling `applyNodeEdit` when editing a tag on a Node and there are no edits to make. (#2161)

17.158.3 Improved Documentation

- Fix typo in docstrings from `$lib.model.tags` Stormtypes. (#2160)

17.159 v2.33.1 - 2021-04-13

17.159.1 Bugfixes

- Fix a regression when expanding list objects in Storm. (#2154)

17.160 v2.33.0 - 2021-04-12

17.160.1 Features and Enhancements

- Add CWE and CVSS support to the `risk:vuln` form. (#2143)
- Add a new Stormtypes library, `$lib.infosec.cvss`, to assist with parsing CVSS data, computing scores, and updating `risk:vuln` nodes. (#2143)
- Add ATT&CK, CWD, and CPE support to the IT model. (#2143)
- Add `it:network`, `it:domain`, `it:account`, `it:group` and `it:login` guid forms to model common IT concepts. (#2096)
- Add a new model, `project`, to model projects, tickets, sprints and epics. The preliminary forms for this model include `proj:project`, `proj:sprint`, `proj:ticket`, `proj:comment`, and `projec:project`. (#2096)
- Add a new Stormtypes library, `$lib.project`, to assist with using the project model. The API is provisional. (#2096)
- Allow lifting `guid` types with the prefix (`^=`) operator. (#2096)

- Add `ou:contest:result:url` to record where to find contest results. (#2144)
- Allow subquery as a value in additional places in Storm. This use must yield exactly one node. Secondary property assignments to array types may yield multiple nodes. (#2137)
- Tighten up Storm iterator behavior on the backend. This should not have have user-facing changes in Storm behavior. (#2148) (#2096)
- Update the Cell backup routine so that it blocks the ioloop less. (#2145)
- Expose the remote name and version of Storm Services in the `service.list` command. (#2149)
- Move test deprecated model elements into their own Coremodule. (#2150)
- Update `lark` dependency. (#2146)

17.160.2 Bugfixes

- Fix incorrect grammer in model.edge commands. (#2147)
- Reduce unit test memory usage. (#2152)
- Pin `jupyter-client` library. (#2153)

17.161 v2.32.1 - 2021-04-01

17.161.1 Features and Enhancements

- The Storm `$lib.exit()` function now takes message arguments similar to `$lib.warn()` and fires that message into the run time as a `warn` prior to stopping the runtime. (#2138)
- Update `pygments` minimum version to `v2.7.4`. (#2139)

17.161.2 Bugfixes

- Do not allow light edge creation on runt nodes. (#2136)
- Fix backup test timeout issues. (#2141)
- Fix the `synapse.lib.msgpack.en()` function so that now raises the correct exceptions when operating in fallback mode. (#2140)
- Fix the `Snap.addNodes()` API handling of deprecated model elements when doing bulk data ingest. (#2142)

17.162 v2.32.0 - 2021-03-30

17.162.1 Features and Enhancements

- Increase the verbosity of logging statements related to Cell backup operations. This allows for better visibility into what is happening while a backup is occurring. (#2124)
- Add Telepath and Storm APIs for setting all the roles of a User at once. (#2127)
- Expose the Synapse package commit hash over Telepath and Stormtypes. (#2133)

17.162.2 Bugfixes

- Increase the process spawn timeout for Cell backup operations. Prevent the Cell backup from grabbing lmdb transactions for slabs in the cell local tmp directory. (#2124)

17.163 v2.31.1 - 2021-03-25

17.163.1 Bugfixes

- Fix a formatting issue preventing Python packages from being uploaded to PyPI. (#2131)

17.164 v2.31.0 - 2021-03-24

17.164.1 Features and Enhancements

- Add initial capability for exporting STIX 2.1 from the Cortex. (#2120)
- Refactor how lift APIs are implemented, moving them up to the Cortex itself. This results in multi-layer lifts now yielding nodes in a sorted order. (#2093) (#2128)
- Add `$lib.range()` Storm function to generate ranges of integers. (#2122)
- Add an `errok` option to the `$lib.time.parse()` Storm function to allow the function to return `$lib.null` if the time string fails to parse. (#2126)
- Don't execute Cron jobs, Triggers, or StormDmons for locked users. (#2123) (#2129)
- The `git` commit hash is now embedded into the `synapse.lib.version` module when building PyPi packages and Docker images. (#2119)

17.164.2 Improved Documentation

- Update Axon wget API documentation to note that we always store the body of the HTTP response, regardless of status code. (#2125)

17.165 v2.30.0 - 2021-03-17

17.165.1 Features and Enhancements

- Add `$lib.trycast()` to allow for Storm control flow based on type normalization. (#2113)

17.165.2 Bugfixes

- Resolve a bug related to pivoting to a secondary property that is an array value. (#2111)
- Fix an issue with Aha and persisting the online state of services upon startup. (#2103)
- Convert the type of `inet:web:acct:singup:client:ipv6` from a `inet:ipv4` to an `inet:ipv6`. (#2114)
- Fix an idempotency issue when deleting a custom form. (#2112)

17.165.3 Improved Documentation

- Update README.rst. (#2115) (#2117) (#2116)

17.166 v2.29.0 - 2021-03-11

This release includes a Cortex storage Layer bugfix. It does an automatic upgrade upon startup to identify and correct invalid array index values. Depending on time needed to perform this automatic upgrade, the Cortex may appear unresponsive. Deployments with startup or liveliness probes should have those disabled while this upgrade is performed to prevent accidental termination of the Cortex process.

17.166.1 Features and Enhancements

- Add a `reverse` argument to `$lib.sorted()` to allow a Storm user to easily reverse an iterable item. (#2109)
- Update minimum required versions of Tornado and PyYAML. (#2108)

17.166.2 Bugfixes

- Fix an issue with Array property type deletion not properly deleting values in the `byarray` index. This requires an automatic data migration done at Cortex startup to remove extra index values which may be present in the index. (#2104) (#2106)
- Fix issues with using the Storm ?= operator with types which can generate multiple values from a given input string when making nodes. (#2105) (#2107)

17.166.3 Improved Documentation

- Add Devops documentation explaining our Docker container offerings. (#2104) (#2110)

17.167 v2.28.1 - 2021-03-08

17.167.1 Bugfixes

- Fix `$lib.model.prop()` API when called with a universal property. It now returns `$lib.null` instead of raising an exception. (#2100)
- Fix the streaming backup API when used with Telepath and SSL. (#2101)

17.167.2 Improved Documentation

- Add API documentation for the Axon. (#2098)
- Update the Storm pivot reference documentation. (#2101)

17.168 v2.28.0 - 2021-02-26

17.168.1 Features and Enhancements

- Add `String.reverse()` Stormtypes API to reverse a string. (#2086)
- Add Cell APIs for streaming compressed backups. (#2084) (#2091)
- Refactor `snap.addNodes()` to reduce the transaction count. (#2087) (#2090)
- Add `$lib.axon.list()` Stormtypes API to list hashes in an Axon. (#2088)
- Add user permissions requirements for Aha CSR signing. (#2089)
- Add `aha:svcinfo` configuration option for the base Cell. (#2089)
- Add interfaces to the output of `model.getModelDefs()` and the `getModelDict()` APIs. (#2092)
- Update pylmdb to `v1.1.1`. (#2076)

17.168.2 Bugfixes

- Fix incorrect permissions check in the `merge --diff` Storm command. (#2085)
- Fix service teardown issue in Aha service on fini. (#2089)
- Fix possible `synapse.tools.cmdr` teardown issue when using Aha. (#2089)
- Cast `synapse_minversion` from Storm Packages into a tuple to avoid packages added with HTTP endpoints from failing to validate. (#2095)

17.168.3 Improved Documentation

- Add documentation for the Aha discovery service. (#2089)
- Add documentation for assigning secondary properties via subquery syntax. (#2097)

17.169 v2.27.0 - 2021-02-16

17.169.1 Features and Enhancements

- Allow property assignment and array operations from subqueries. (#2072)
- Add APIs to the Axon to allow the deletion of blobs via Telepath and HTTP APIs. (#2080)
- Add a `str.slice()` stormtypes method to allow easy string slicing. (#2083)
- Modularize the Storm HTTP API handlers. (#2082)

17.169.2 Bugfixes

- Fix Agenda events which were not being properly tracked via the Nexus. (#2078)

17.169.3 Improved Documentation

- Add documentation for the Cortex `/api/v1/storm/export` HTTP endpoint. This also included documentation for the scrub option in Storm. (#2079)
- Add a Code of Conduct for Synapse. (#2081)

17.170 v2.26.0 - 2021-02-05

17.170.1 Features and Enhancements

- Add Storm commands for easily adding, deleting, and listing layer push and pull configurations. (#2071)

17.170.2 Bugfixes

- Fix `layer.getPropCount()` API for universal properties. (#2073)
- Add a missing async yield in `Snap.addNodes()`. (#2074)
- Constrain lmdb version due to unexpected behavior in `v1.1.0`. (#2075)

17.170.3 Improved Documentation

- Update user docs for Storm flow control and data model references. (#2066)

17.171 v2.25.0 - 2021-02-01

17.171.1 Features and Enhancements

- Implement tag model based pruning behavior for controlling how individual tag trees are deleted from nodes. (#2067)
- Add model interfaces for defining common sets of properties for forms, starting with some file mime metadata. (#2040)
- Add `file:mime:msdoc`, `file:mime:msxls`, `file:mime:msppt`, and `file:mime:rtf` forms. (#2040)
- Tweak the ival normalizer to auto-expand intervals with a single element. (#2070)
- Removed the experimental `spawn` feature of the Storm runtime. (#2068)

17.171.2 Bugfixes

- Add a missing async yield statement in `View.getEdgeVerbs()`. (#2069)

17.171.3 Improved Documentation

- Correct incorrect references to the `synapse.tools.easycert` documentation. (#2065)

17.172 v2.24.0 - 2021-01-29

17.172.1 Features and Enhancements

- Add support for storing model metadata for tags and support for enforcing tag trees using regular expressions. (#2056)
- Add `ou:contest:url` secondary property. (#2059)
- Add `synapse.lib.autodoc` to collect some Storm documentation helpers into a single library. (#2034)
- Add `tag.prune` Storm command to remove parent tags when removing a leaf tag from a node. (#2062)
- Update the `msgpack` Python dependency to version `v1.0.2`. (#1735)
- Add logs to Cell backup routines. (#2060)
- Export the Layer iterrows APIs to the CoreApi. (#2061)

17.172.2 Bugfixes

- Do not connect to Aha servers when they are not needed. (#2058)
- Make the array property `ou:org:industries` a unique array property. (#2059)
- Add permission checks to the Storm `movetag` command. (#2063)
- Add permissions checks to the Storm `edges.del` command. (#2064)

17.172.3 Improved Documentation

- Add documentation for the `synapse.tools.genpkg` utility, for loading Storm packages into a Cortex. (#2057)
- Refactor the Stormtypes documentation generation to make it data driven. (#2034)

17.173 v2.23.0 - 2021-01-21

17.173.1 Features and Enhancements

- Add support for ndef based light edge definitions in the `syn.nodes` feed API. (#2051) (#2053)
- Add ISIC codes to the `ou:industry` form. (#2054) (#2055)
- Add secondary properties `:loc`, `:latlong`, and `:place` to the `inet:web:action` and `inet:web:logon` forms. (#2052)

- Add secondary property `:enabled` to the form `it:app:yara:rule`. (#2052)
- Deprecate the `file:string` and `ou:member` forms, in favor of using light edges for storing those relationships. (#2052)

17.174 v2.22.0 - 2021-01-19

17.174.1 Features and Enhancements

- Allow expression statements to be used in Storm filters. (#2041)
- Add `file:subfile:path` secondary property to record the path a file was stored in a parent file. The corresponding `file:subfile:name` property is marked as deprecated. (#2043)
- Make the Axon `wget()` timeout a configurable parameter. (#2047)
- Add a `Cortex.exportStorm()` on the Cortex which allows for exporting nodes from a Storm query which can be directly ingested with the `syn.nodes` feed function. If the data is serialized using msgpack and stored in a Axon, it can be added to a Cortex with the new `Cortex.feedFromAxon()` API. A new HTTP API, `/api/v1/storm/export`, can be used to get a msgpacked file using this export interface. (#2045)

17.174.2 Bugfixes

- Fix issues in the Layer push and pull loop code. (#2044) (#2048)
- Add missing `toprim()` and `tostr()` calls for the Stormtypes Whois guid generation helpers. (#2046)
- Fix behavior in the Storm lookup mode which failed to lookup some expected results. (#2049)
- Fix `$lib.pkg.get()` return value when the package is not present. (#2050)

17.175 v2.21.1 - 2021-01-04

17.175.1 Bugfixes

- Fix a variable scoping issue causing a race condition. (#2042)

17.176 v2.21.0 - 2020-12-31

17.176.1 Features and Enhancements

- Add a Storm `wget` command which will download a file from a URL using the Cortex Axon and yield `inet:urlfile` nodes. (#2035)
- Add a `--diff` option to the `merge` command to enumerate changes. (#2037)
- Allow StormLib Layer API to dynamically update a Layer's logedits setting. (#2038)
- Add StormLib APIs for adding and deleting extended model properties, forms and tag properties. (#2039)

17.176.2 Bugfixes

- Fix an issue with the JsonStor not created nested entries properly. (#2036)

17.177 v2.20.0 - 2020-12-29

17.177.1 Features and Enhancements

- Correct the StormType `Queue.pop()` API to properly pop and return only the item at the specified index or the next entry in the Queue. This simplifies the intent behind the `.pop()` operation; and removes the `cull` and `wait` parameters which were previously on the method. (#2032)

17.177.2 Bugfixes

- Use `resp.iter_chunked` in the Axon `.wget()` API to improve compatibility with some third party libraries. (#2030)
- Require the use of a msgpack based deepcopy operation in handling storage nodes. (#2031)
- Fix for ambiguous whitespace in Storm command argument parsing. (#2033)

17.178 v2.19.0 - 2020-12-27

17.178.1 Features and Enhancements

- Add APIs to remove decommissioned services from AHA servers.
- Add (optional) explicit network parameters to AHA APIs. (#2029)
- Add cell.isCellActive() API to differentiate leaders/mirrors. (#2028)
- Add pop() method to Storm list objects. (#2027)

17.178.2 Bugfixes

- Fix bug in dry-run output of new merge command. (#2026)

17.179 v2.18.1 - 2020-12-24

17.179.1 Bugfixes

- Make syncIndexEvents testing more resiliant
- Make syncIndexEvents yield more often when filtering results (#2025)
- Update push/pull tests to use new waittask() API
- Raise clear errors in ambiguous use of node.tagglobs() API
- Update model docs and examples for geo:latitude and geo:longitude

- Support deref form names in storm node add expressions (#2024)
- Update tests to normalize equality comparison values (#2023)

17.180 v2.18.0 - 2020-12-23

17.180.1 Features and Enhancements

- Added axon.size() API and storm plumbing (#2020)

17.180.2 Bugfixes

- Fix active coro issue uncovered with cluster testing (#2021)

17.181 v2.17.1 - 2020-12-22

17.181.1 Features and Enhancements

- Added (BETA) RST pre-processor to embed Storm output into RST docs. (#1988)
- Added a `merge` command to allow per-node Layer merge operations to be done. (#2009)
- Updated storm package format to include a semver version string. (#2016)
- Added telepath proxy getPipeline API to minimize round-trip delay. (#1615)
- Added Node properties iteration and setitem APIs to storm. (#2011)

17.181.2 Bugfixes

- Fixes for active coro API and internal layer API name fixes. (#2018)
- Allow :prop -+> * join syntax. (#2015)
- Make getFormCount() API return a primitive dictionary. (#2014)
- Make StormVarListError messages more user friendly. (#2013)

17.182 v2.17.0 - 2020-12-22

`2.17.0` was not published due to CI issues.

17.183 v2.16.1 - 2020-12-17

17.183.1 Features and Enhancements

- Allow the `matchdef` used in the `Layer.syncIndexEvents()` API to match on tagprop data. (#2010)

17.183.2 Bugfixes

- Properly detect and raise a client side exception in Telepath generators when the underlying Link has been closed. (#2008)
- Refactor the Layer push/push test to not reach through the Layer API boundary. (#2012)

17.183.3 Improved Documentation

- Add documentation for Storm raw pivot syntax. (#2007)
- Add documentation for recently added Storm commands. (#2007)
- General cleanup and clarifications. (#2007)

17.184 v2.16.0 - 2020-12-15

17.184.1 Features and Enhancements

- Replaced the View sync APIs introduced in `v2.14.0` with Layer specific sync APIs. (#2003)
- Add `$lib.regex.matches()` and `$lib.regex.search()` Stormtypes APIs for performing regular expression operations against text in Storm. (#1999) (#2005)
- Add `synapse.tools.genpkg` for generating Storm packages and loading them into a Cortex. (#2004)
- Refactored the StormDmon implementation to use a single async task and allow the Dmons to be restarted via `$lib.dmon.bump(iden)`. This replaces the outer task / inner task paradigm that was previously present. Also add the ability to persistently disable and enable a StomDmon. (#1998)
- Added `aha://` support to the `synapse.tools.pushfile` and `synapse.tools.pullfile` tools. (#2006)

17.184.2 Bugfixes

- Properly handle whitespace in keyword arguments when calling functions in Storm. (#1997)
- Fix some garbage collection issues causing periodic pauses in a Cortex due to failing to close some generators used in the Storm Command AST node. (#2001) (#2002)
- Fix scope based permission checks in Storm. (#2000)

17.185 v2.15.0 - 2020-12-11

17.185.1 Features and Enhancements

- Add two new Cortex APIs: `syncIndexEvents` and `syncLayerEvents` useful for external indexing. (#1948) (#1996)
- LMDB Slab improvements: Allow dupfixed dbs, add `firstkey` method, inline `_ispo2`, add HotCount deletion. (#1948)
- Add method to merge sort sorted async generators. (#1948)

17.185.2 Bugfixes

- Ensure parent FQDN exists even in out-of-order node edit playback. (#1995)

17.186 v2.14.2 - 2020-12-10

17.186.1 Bugfixes

- Fix an issue with the new layer push / pull code. (#1994)
- Fix an issue with the url sanitization function when the path contains an `@` character. (#1993)

17.187 v2.14.1 - 2020-12-09

17.187.1 Features and Enhancements

- Add a `/api/v1/active` HTTP API to the Cell that can be used as an unauthenticated liveliness check. (#1987)
- Add `$lib.pip.gen()` Stormtypes API for ephemeral queues and bulk data access in Storm. (#1986)
- Add a `$lib.model.tagprop()` Stormtypes API for retrieving Tagprop definitions. (#1990)
- Add efficient View and Layer push/pull configurations. (#1991) (#1992)
- Add `getAhaUrls()` to the Aha service to prepare for additional service discovery. (#1989)
- Add a `/api/v1/auth/onepass/issue` HTTP API for an admin to mint a one-time password for a Cell user. (#1982)

17.187.2 Bugfixes

- Make `aha://` urls honor local paths. (#1985)

17.188 v2.14.0 - 2020-12-09

`2.14.0` was not published due to CI issues.

17.189 v2.13.0 - 2020-12-04

17.189.1 Features and Enhancements

- Add `$lib.pkg.get()` StormTypes function to get the Storm Package definition for a given package by name. (#1983)

17.189.2 Bugfixes

- The user account provisioned by the `aha:admin` could be locked out. Now, upon startup, if they have been locked out or had their admin status removed, they are unlocked and admin is reset. (#1984)

17.190 v2.12.3 - 2020-12-03

17.190.1 Bugfixes

- Prevent OverflowError exceptions which could have resulted from lift operations with integer storage types. (#1980)
- Remove `inet:ipv4` norm routine wrap-around behavior for integers which are outside the normal bounds of IPv4 addresses. (#1979)
- Fix `view.add` and fork related permissions. (#1981)
- Read `telepath.yaml` when using the `synapse.tools.cellauth` tool. (#1981)

17.191 v2.12.2 - 2020-12-01

This release also includes the changes from v2.12.1, which was not released due to an issue with CI pipelines.

17.191.1 Bugfixes

- Add the missing API `getPathObjs` on the JsonStorCell. (#1976)
- Fix the HasRelPropCond AST node support for Storm pivprop operations. (#1972)
- Fix support for the `aha:registry` config parameter in a Cell to support an array of strings. (#1975)
- Split the `Cortex.addForm()` Nexus handler into two parts to allow for safe event replay. (#1978)
- Stop forking a large number of child layers in a View persistence test. (#1977)

17.192 v2.12.1 - 2020-12-01

17.192.1 Bugfixes

- Add the missing API `getPathObjs` on the JsonStorCell. (#1976)
- Fix the HasRelPropCond AST node support for Storm pivprop operations. (#1972)
- Fix support for the `aha:registry` config parameter in a Cell to support an array of strings. (#1975)

17.193 v2.12.0 - 2020-11-30

17.193.1 Features and Enhancements

- Add a `onload` paramter to the `stormpkg` definition. This represents a Storm query which is executed every time the `stormpkg` is loaded in a Cortex. (#1971) (#1974)
- Add the ability, in Storm, to unset variables, remove items from dictionaries, and remove items from lists. This is done via assigning `$lib.undef` to the value to be removed. (#1970)
- Add support for SOCKS proxy support for outgoing connections from an Axon and Cortex, using the `'http:proxy` configuration option. This configuration value must be a valid string for the `aiohttp_socks.ProxyConnector.from_url()` API. The SOCKS proxy is used by the Axon when downloading files; and by the Cortex when making HTTP connections inside of Storm. (#1968)
- Add `aha:admin` to the Cell configuration to provide a common name that is used to create an admin user for remote access to the Cell via the Aha service. (#1969)
- Add `auth:ctor` and `auth:conf` config to the Cell in order to allow hooking the construction of the `HiveAuth` object. (#1969)

17.194 v2.11.0 - 2020-11-25

17.194.1 Features and Enhancements

- Optimize Storm lift and filter queries, so that more efficient lift operations may be performed in some cases. (#1966)
- Add a `Axon.wget()` API to allow the Axon to retrieve files directly from a URL. (#1965)
- Add a JsonStor Cell, which allows for hierarchical storage and retrieval of JSON documents. (#1954)
- Add a Cortex HTTP API, `/api/v1/storm/call`. This behaves like the `CoreApi.callStorm()` API. (#1967)
- Add `:client:host` and `:server:host` secondary properties to the `inet:http:request` form. (#1955)
- Add `:host` and `:acct` secondary properties to the `inet:search:query` form. (#1955)
- Add a Telepath service discovery implementation, the Aha cell. The Aha APIs are currently provisional and subject to change. (#1954)

17.195 v2.10.2 - 2020-11-20

17.195.1 Features and Enhancements

- The Storm `cron.at` command now supports a `--now` flag to create a cron job which immediately executes. (#1963)

17.195.2 Bugfixes

- Fix a cleanup race that caused occasional `test_lmdbslab_base` failures. (#1962)
- Fix an issue with `EDIT_NODEDATA_SET` nodeedits missing the `oldv` value. (#1961)
- Fix an issue where `cron.cleanup` could have prematurely deleted some cron jobs. (#1963)

17.196 v2.10.1 - 2020-11-17

17.196.1 Bugfixes

- Fix a CI issue which prevented the Python `sdist` package from being uploaded to PyPi. (#1960)

17.197 v2.10.0 - 2020-11-17

17.197.1 Announcements

The `v2.10.0` Synapse release contains support for Python 3.8. Docker images are now built using a Python 3.8 image by default. There are also Python 3.7 images available as `vertexproject/synapse:master-py37` and `vertexproject/synapse:v2.x.x-py37`.

17.197.2 Features and Enhancements

- Python 3.8 release support for Docker and PyPi. (#1921) (#1956)
- Add support for adding extended forms to the Cortex. This allows users to define their own forms using the existing types which are available in the Synapse data model. (#1944)
- The Storm `and` and `or` statements now short-circuit and will return when their logical condition is first met. This means that subsequent clauses in those statements may not be executed. (#1952)
- Add a mechanism for Storm Services to specify commands which may require privilege elevation to execute. An example of this may be to allow a command to create nodes; without managning individual permissions on what nodes a user may normally be allowed to create. Services using this mechanism wiill use the `storm.asroot.cmd.<<cmd name>>` hierarchy to grant this permission. (#1953) (#1958)
- Add `$lib.json` Stormtypes Library to convert between string data and primitives. (#1949)
- Add a `parallel` command to allow for executing a portion of a Storm query in parallel. Add a `background` command to execute a Storm query as a detached task from the current query, capturing variables in the process. (#1931) (#1957)
- Add a `$lib.exit()` function to StormTypes to allow for quickly exiting a Storm query. (#1931)

- Add `$lib.bytes.upload()` to Stormtypes for streaming bytes into the Axon that the Cortex is configured with. (#1945)
- Add Storm commands to manage locking and unlocking deprecated model properties. (#1909)
- Add `cron.cleanup` command to make it easy to clean up completed cron jobs. (#1942)
- Add date of death properties and consistently named photo secondary properties. (#1929)
- Add model additions for representing education and awards. (#1930)
- Add additional account linkages to the `inet` model for users and groups. (#1946)
- Add `inet:web:hashtag` as its own form, and add `:hashtags` to `inet:web:post`. (#1946)
- Add `lang:translation` to capture language translations of texts in a more comprehensive way than older `lang` model forms did. The `lang:idiom` and `lang:trans` forms have been marked as deprecated. (#1946)
- Update the `ou` model to add `ou:attendee` and `ou:contest` and `ou:contest:result` forms. Several secondary properties related to conference attendance have been marked deprecated. (#1946)
- The `ps:persona` and `ps:persona:has` forms have been marked as deprecated. (#1946)
- Add `ps:contactlist` to allow collecting multiple `ps:contact` nodes together. (#1935)
- Allow the Storm Service cmdargs to accept any valid model type in the `type` value. (#1923) (#1936)
- Add >, <, >= and <= comparators for `inet:ipv4` type. (#1938)
- Add configuration options to the Axon to limit the amount of data which can be stored in it. Add a configuration option the Cortex to limit the number of nodes which may be stored in a given Cortex. (#1950)

17.197.3 Bugfixes

- Fix a potential incorrect length for Spooled sets during fallback. (#1937)
- Fix an issue with the Telepath `Client` object caching their `Method` and `GenrMethod` attributes across re-connections of the underlying `Proxy` objects. (#1939) (#1941)
- Fix a bug where a temporary spool slab cleanup failed to remove all files from the filesystem that were created when the slab was made. (#1940)
- Move exceptions which do not subclass `SynErr` out of `synapse/exc.py`. (#1947) (#1951)

17.198 v2.9.2 - 2020-10-27

17.198.1 Bugfixes

- Fix an issue where a Cortex migrated from a *01x* release could overwrite entries in a Layer's historical nodeedit log. (#1934)
- Fix an issue with the layer definition schema. (#1927)

17.199 v2.9.1 - 2020-10-22

17.199.1 Features and Enhancements

- Reuse existing an existing `DateTime` object when making time strings. This gives a slight performance boost for the `synapse.lib.time.repr()` function. (#1919)
- Remove deprecated use of `loop` arguments when calling `asyncio` primitives. (#1920)
- Allow Storm Services to define a minimum required Synapse version by the Cortex. If the Cortex is not running the minimum version, the Cortex will not load (#1900)
- Only get the nxsindx in the `Layer.storeNodeEdits()` function if logging edits. (#1926)
- Include the Node iden value in the `CantDelNode` exception when attempting to delete a Node failes due to existing references to the node. (#1926)
- Take advantage of the LMDB append operation when possible. (#1912)

17.199.2 Bugfixes

- Fix an issues in the Telepath Client where an exception thrown by a onlink function could cause additional linkloop tasks to be spawned. (#1924)

17.200 v2.9.0 - 2020-10-19

17.200.1 Announcements

The `v2.9.0` Synapse release contains an automatic Cortex Layer data migration. The updated layer storage format reduces disk and memory requirements for a layer. It is recommended to test this process with a backup of a Cortex before updating a production Cortex.

In order to maximize the space savings from the new layer storage format, after the Cortex has been migrated to `v2.9.0`, one can take a cold backup of the Cortex and restore the Cortex from that backup. This compacts the LMDB databases which back the Layers and reclaims disk space as a result. This is an optional step; as LMDB will eventually re-use the existing space on disk.

If there are any questions about this, please reach out in the Synapse Slack channel so we can assist with any data migration questions.

17.200.2 Features and Enhancements

- Optimize the layer storage format for memory size and performance. (#1877) (#1885) (#1899) (#1917)
- Initial support Python 3.8 compatibility for the core Synapse library. Additional 3.8 support (such as wheels and Docker images) will be available in future releases. (#1907)
- Add a read only Storm option to the Storm runtime. This option prevents executing commands or Stormtypes functions which may modify data in the Cortex. (#1869) (#1916)
- Allow the Telepath Dmon to disconnect clients using a ready status. (#1881)
- Ensure that there is only one online backup of a Cell occurring at a time. (#1883)

- Added `.lower()`, `.strip()`, `.lstrip()` and `.rstrip()` methods to the Stormtypes Str object. These behave like the Python `str` methods. (#1886) (#1906)
- When scraping text, defanged indicators are now refanged by default. (#1888)
- Normalize read-only property declarations to use booleans in the data model. (#1887)
- Add `lift.byverb` command to allow lifting nodes using a light edge verb. (#1890)
- Add netblock and range lift helpers for `inet:ipv6` type, similar to the helpers for `inet:ipv4`. (#1869)
- Add a `edges.del` command to bulk remove light weight edges from nodes. (#1893)
- The `yield` keyword in Storm now supports iterating over Stormtypes List and Set objects. (#1898)
- Add `ou:contract`, `ou:industry` and `it:reveng:function:strings` forms to the data model. (#1894)
- Add some display type-hinting to the data model for some string fields which may be multi-line fields. (#1892)
- Add `getFormCounts()` API to the Stormtypes View and Layer objects. (#1903)
- Allow Cortex layers to report their total size on disk. This is exposed in the Stormtypes `Layer.pack()` method for a layer. (#1910)
- Expose the remote Storm Service name in the `$lib.service.get()` Stormtypes API. This allows getting a service object without knowing the name of the service as it was locally added to a Cortex. Also add a `$lib.service.has()` API which allows checking to see if a service is available on a Cortex. (#1908) (#1915)
- Add regular expression (`~=`) and prefix matching (`^=`) expression comparators that can be used with logical expressions inside of Storm. (#1906)
- Promote `CoreApi.addFeedData()` calls to tracked tasks which can be viewed and terminated. (#1918)

17.200.3 Bugfixes

- Fixed a Storm bug where attempting to access an undeclared variable silently fails. This will now raise a `NoSuchVar` exception. This is verified at runtime, not at syntax evaluation. (#1916)
- Ensure that Storm HTTP APIs tear down the runtime task if the remote disconnects before consuming all of the messages. (#1889)
- Fix an issue where the `model.edge.list` command could block the ioloop for large Cortex. (#1890)
- Fix a regex based lifting bug. (#1899)
- Fix a few possibly greedy points in the AST code which could have resulted in greedy CPU use. (#1902)
- When pivoting across light edges, if the destination form was not a valid form, nothing happened. Now a StormRuntimeError is raised if the destination form is not valid. (#1905)
- Fix an issue with spawn processes accessing lmdb databases after a slab resize event has occurred by the main process. (#1914)
- Fix a slab teardown race seen in testing Python 3.8 on MacOS. (#1914)

17.200.4 Deprecations

- The `0.1.x` to `2.x.x` Migration tool and associated Cortex sync service has been removed from Synapse in the `2.9.0` release.

17.200.5 Improved Documentation

- Clarify user documentation for pivot out and pivot in operations. (#1891)
- Add a deprecation policy for Synapse Data model elements. (#1895)
- Pretty print large data structures that may occur in the data model documentation. (#1897)
- Update Storm Lift documentation to add the `?=` operator. (#1904)

17.201 v2.8.0 - 2020-09-22

17.201.1 Features and Enhancements

- Module updates to support generic organization identifiers, generic advertising identifiers, asnet6 and a few other secondary property additions. (#1879)
- Update the Cell backup APIs to perform a consistent backup across all slabs for a Cell. (#1873)
- Add support for a environment variable, `SYN_LOCKMEM_DISABLE` which will disable any memory locking of LMDB slabs. (#1882)

17.201.2 Deprecations

- The `0.1.x` to `2.x.x` Migration tool and and associated Cortex sync service will be removed from Synapse in the `2.9.0` release. In order to move forward to `2.9.0`, please make sure that any Cortexes which still need to be migrated will first be migrated to `2.8.x` prior to attempting to use `2.9.x`.

17.201.3 Improved Documentation

- Add Synapse README content to the Pypi page. This was a community contribution from https://github.com/wesinator. (#1872)

17.202 v2.7.3 - 2020-09-16

17.202.1 Deprecations

- The `0.1.x` to `2.x.x` Migration tool and and associated Cortex sync service will be removed from Synapse in the `2.9.0` release. In order to move forward to `2.9.0`, please make sure that any Cortexes which still need to be migrated will first be migrated to `2.8.x` prior to attempting to use `2.9.x`. (#1880)

17.202.2 Bugfixes

- Remove duplicate words in a comment. This was a community contribution from enadjoe. (#1874)
- Fix a nested Nexus log event in Storm Service deletion. The `del` event causing Storm code execution could lead to nested Nexus events, which is incongruent with how Nexus change handlers work. This now spins off the Storm code in a free-running coroutine. This does change the service `del` semantics since any support Storm packages a service had may be removed by the time the handler executes. (#1876)
- Fix an issue where the `cull` parameter was not being passed to the multiqueue properly when calling `.gets()` on a Storm Types Queue object. (#1876)
- Pin the `nbconvert` package to a known working version, as `v6.0.0` of that package broke the Synapse document generation by changing how templates work. (#1876)
- Correct `min` and `max` integer examples in tagprop documentation and tests. (#1878)

17.203 v2.7.2 - 2020-09-04

17.203.1 Features and Enhancements

- Update tests for additional test code coverage. This was a community contribution from blackout. (#1867)
- Add implicit links to documentation generated for Storm services, to allow for direct linking inside of documentation to specific Storm commands. (#1866)
- Add future support for deprecating model elements in the Synapse data model. This support will produce client and server side warnings when deprecated model elements are used or loaded by custom model extensions or CoreModules. (#1863)

17.203.2 Bugfixes

- Update `FixedCache.put()` to avoid a cache miss. This was a community contribution from blackout. (#1868)
- Fix the ioloop construction to be aware of `SYN_GREEDY_CORO` environment variable to put the ioloop into debug mode and log long-running coroutines. (#1870)
- Fix how service permissions are checked in `$lib.service.get()` and `$lib.service.wait()` Storm library calls. These APIs now first check `service.get.<service iden>` before checking `service.get.<service name>` permissions. A successful `service.get.<service name>` check will result in a warning to the client and the server. (#1871)

17.204 v2.7.1 - 2020-08-26

17.204.1 Features and Enhancements

- Refactor an Axon unit test to make it easier to test alternative Axon implementations. (#1862)

17.204.2 Bugfixes

- Fix an issue in `synapse.tools.cmdr` where it did not ensure that the users Synapse directory was created before trying to open files in the directory. (#1860) (#1861)

17.204.3 Improved Documentation

- Fix an incorrect statement in our documentation about the intrinsic Axon that a Cortex creates being remotely accessible. (#1862)

17.205 v2.7.0 - 2020-08-21

17.205.1 Features and Enhancements

- Add Telepath and HTTP API support to set and remove global Storm variables. (#1846)
- Add Cell level APIs for performing the backup of a Cell. These APIs are exposed inside of a Cortex via a Storm Library. (#1844)
- Add support for Cron name and doc fields to be editable. (#1848)
- Add support for Runtime-only (`runt`) nodes in the PivotOut operation (`-> *`). (#1851)
- Add `:nicks` and `:names` secondary properties to `ps:person` and `ps:persona` types. (#1852)
- Add a new `ou:position` form and a few associated secondary properties. (#1849)
- Add a step to the CI build process to smoke test the sdist and wheel packages before publishing them to PyPI. (#1853)
- Add support for representing `nodedata` in the command hinting for Storm command implementations and expose it on the `syn:cmd` runt nodes. (#1850)
- Add package level configuration data to Storm Packages in the `modconf` value of a package definition. This is added to the runtime variables when a Storm package is imported, and includes the `svciden` for packages which come from Storm Services. (#1855)
- Add support for passing HTTP params when using `$lib.inet.http.*` functions to make HTTP calls in Storm. (#1856)
- Log Storm queries made via the `callStorm()` and `count()` APIs. (#1857)

17.205.2 Bugfixes

- Fix an issue were some Storm filter operations were not yielding CPU time appropriately. (#1845)

17.205.3 Improved Documentation

- Remove a reference to deprecated `eval()` API from quickstart documentation. (#1858)

17.206 v2.6.0 - 2020-08-13

17.206.1 Features and Enhancements

- Support `+hh:mm` and `+hh:mm` timezone offset parsing when normalizing `time` values. (#1833)
- Enable making mirrors of Cortex mirrors work. (#1836)
- Remove read-only properties from `inet:flow` and `inet:http:request` forms. (#1840)
- Add support for setting nodedata and light edges in the `syn.nodes` ingest format. (#1839)
- Sync the LMDB Slab replay log if it gets too large instead of waiting for a force commit operation. (#1838)
- Make the Agenda unit tests an actual component test to reduce test complexity. (#1837)
- Support glob patterns when specifying files to upload to an Axon with `synapse.tools.pushfile`. (#1837)
- Use the node edit metadata to store and set the `.created` property on nodes, so that mirrors of Cortexes have consistent `.created` timestamps. (#1765)
- Support parent runtime variables being accessed during the execution of a `macro.exec` command. (#1841)
- Setting tags from variable values in Storm now calls `s_stormtypes.tostr()` on the variable value. (#1843)

17.206.2 Bugfixes

- The Storm `tree` command now catches the Synapse `RecursionLimitHit` error and raises a `StormRuntimeError` instead. The `RecursionLimitHit` being raised by that command was, in practice, confusing. (#1832)
- Resolve memory leak issues related to callStorm and Base object teardowns with exceptions. (#1842)

17.207 v2.5.1 - 2020-08-05

17.207.1 Features and Enhancements

- Add performance oriented counting APIs per layer, and expose them via Stormtypes. (#1813)
- Add the ability to clone a layer, primarily for benchmarking and testing purposes. (#1819)
- Update the benchmark script to run on remote Cortexes. (#1829)

17.207.2 Bugfixes

- Sanitize passwords from Telepath URLs during specific cases where the URL may be logged. (#1830)

17.207.3 Improved Documentation

- Fix a few typos in docstrings. (#1831)

17.208 v2.5.0 - 2020-07-30

17.208.1 Features and Enhancements

- Refactor the Nexus to remove leadership awareness. (#1785)
- Add support for client-side certificates in Telepath for SSL connections. (#1785)
- Add multi-dir support for CertDir. (#1785)
- Add a `--no-edges` option to the Storm `graph` command. (#1805)
- Add `:doc:url` to the `syn:tag` form to allow recording a URL which may document a tag. (#1805)
- Add `CoreApi.reqValidStorm()` and a `/api/v1/reqvalidstorm` Cortex HTTP API endpoint to validate that a given Storm query is valid Storm syntax. (#1806)
- Support Unicode white space in Storm. All Python *s* (Unicode white space + ASCII separators) is now treated as white space in Storm. (#1812)
- Refactor how StormLib and StormPrim objects access their object locals, and add them to a global registry to support runtime introspection of those classes. (#1804)
- Add smoke tests for the Docker containers built in CircleCI, as well as adding Docker healthchecks to the Cortex, Axon and Cryotank images. (#1815)
- Initialize the names of the default view and layer in a fresh Cortex to `default`. (#1814)
- Add HTTP API endpoints for the Axon to upload, download and check for the existend of files. (#1817) (#1822) (#1824) (#1825)
- Add a `$lib.bytes.has()` API to check if the Axon a Cortex is configured with knows about a given sha256 value. (#1822)
- Add initial model for prices, currences, securities and exchanges. (#1820)
- Add a `:author` field to the `it:app:yara:rule` form. (#1821)
- Add an experimental option to set the NexusLog as a `map_async` slab. (#1826)
- Add an initial transportation model. (#1816)
- Add the ability to dereference an item, from a list of items, in Storm via index. (#1827)
- Add a generic `$lib.inet.http.request()` Stormlib function make HTTP requests with arbitrary verbs. (#1828)

17.208.2 Bugfixes

- Fix an issue with the Docker builds for Synapse where the package was not being installed properly. (#1815)

17.208.3 Improved Documentation

- Update documentation for deploying Cortex mirrors. (#1811)
- Add automatically generated documentation for all the Storm `$lib...` functions and Storm Primitive types. (#1804)
- Add examples of creating a given Form to the automatically generated documentation for the automatically generated datamodel documentation. (#1818)
- Add additional documentation for Cortex automation. (#1797)
- Add Devops documentation for the list of user permissions relevant to a Cell, Cortex and Axon. (#1823)

17.209 v2.4.0 - 2020-07-15

17.209.1 Features and Enhancements

- Update the Storm `scrape` command to make `refs` light edges, instead of `edge:refs` nodes. (#1801) (#1803)
- Add `:headers` and `:response:headers` secondary properties to the `inet:http:request` form as Array types, so that requests can be directly linked to headers. (#1800)
- Add `:headers` secondary property to the `inet:email:messaage` form as Array types, so that messages can be directly linked to headers. (#1800)
- Add additional model elements to support recording additional data for binary reverse engineering. (#1802)

17.210 v2.3.1 - 2020-07-13

17.210.1 Bugfixes

- Prohibit invalid rules from being set on a User or Role object. (#1798)

17.211 v2.3.0 - 2020-07-09

17.211.1 Features and Enhancements

- Add `ps.list` and `ps.kill` commands to Storm, to allow introspecting the runtime tasks during (#1782)
- Add an `autoadd` mode to Storm, which will extract basic indicators and make nodes from them when executed. This is a superset of the behavior in the `lookup` mode. (#1795)
- Support skipping directories in the `synapse.tools.backup` tool. (#1792)
- Add prefix based lifting to the Hex type. (#1796)

17.211.2 Bugfixes

- Fix an issue for prop pivot out syntax where the source data is an array type. (#1794)

17.211.3 Improved Documentation

- Add Synapse data model background on light edges and update the Storm data modification and pivot references for light edges. (#1784)
- Add additional terms to the Synapse glossary. (#1784)
- Add documentation for additional Storm commands. (#1784)
- Update documentation for Array types. (#1791)

17.212 v2.2.2 - 2020-07-03

17.212.1 Features and Enhancements

- Add some small enhancements to the Cortex benchmarking script. (#1790)

17.212.2 Bugfixes

- Fix an error in the help for the `macro.del` command. (#1786)
- Fix rule indexing for the `synapse.tools.cellauth` tool to correctly print the rule offsets. (#1787)
- Remove extraneous output from the Storm Parser output. (#1789)
- Rewrite the language (and private APIs) for the Storm `model.edge` related commands to remove references to extended properties. That was confusing language which was unclear for users. (#1789)
- During 2.0.0 migrations, ensure that Cortex and Layer idens are unique; and make minimum 0.1.6 version requirement for migration. (#1788)

17.213 v2.2.1 - 2020-06-30

17.213.1 Bugfixes

- The Axon test suite was missing a test for calling `Axon.get()` on a file it did not have. This is now included in the test suite. (#1783)

17.213.2 Improved Documentation

- Improve Synapse devops documentation hierarchy. Add note about Cell directories being persistent. (#1781)

17.214 v2.2.0 - 2020-06-26

17.214.1 Features and Enhancements

- Add a `postAnit()` callback to the `synapse.lib.base.Base()` object which is called *after* the `__anit__()` call chain is completed, but before `Base.anit()` returns the object instance to the caller. This is used by the Cell to defer certain Nexus actions until the Cell has completed initializing all of its instance attributes. (#1768)
- Make `synapse.lib.msgpack.en()` raise a `SynErr.NotMsgpackSafe` exception instead of passing through the exception raised by msgpack. (#1768)

17.214.2 Bugfixes

- Add a missing `toprim()` call in `$lib.globals.set()`. (#1778)
- Fix an issue in the quickstart documentation related to permissions. Thank you `enadjoe` for your contribution. (#1779)
- Fix an Cell/Cortex startup issue which caused errors when starting up a Cortex when the last Nexus event was replayed. This has a secondary effect that Cell implementers cannot be making Nexus changes during the `__anit__` methods. (#1768)

17.214.3 Improved Documentation

- Add a minimal Storm Service example to the developer documentation. (#1776)
- Reorganize the Synapse User Guide into a more hierarchical format. (#1777)
- Fill out additional glossary items. (#1780)

17.215 v2.1.2 - 2020-06-18

17.215.1 Bugfixes

- Disallow command and bare string contensts from starting with `//` and `/*` in Storm syntax. (#1769)

17.216 v2.1.1 - 2020-06-16

17.216.1 Bugfixes

- Fix an issue in the autodoc tool which failed to account for Storm Service commands without cmdargs. (#1775)

17.217 v2.1.0 - 2020-06-16

17.217.1 Features and Enhancements

- Add information about light edges to graph carving output. (#1762)
- Add a `geo:json` type and `geo:place:geojson` property to the model. (#1759)
- Add the ability to record documentation for light edges. (#1760)
- Add the ability to delete and set items inside of a MultiQueue. (#1766)

17.217.2 Improved Documentation

- Refactor `v2.0.0` changelog documentation. (#1763)
- Add Vertex branding to the Synapse documentation. (#1767)
- Update Backups documentation in the Devops guide. (#1764)
- Update the autodoc tool to generate documentation for Cell confdefs and StormService information. (#1772)
- Update to separate the devops guides into distinct sections. (#1772)
- Add documentation for how to do boot-time configuration for a Synapse Cell. (#1772)
- Remove duplicate information about backups. (#1774)

17.218 v2.0.0 - 2020-06-08

Initial 2.0.0 release.

17.219 API Deprecation Notice - 2023-10-01

It's time to shed some long standing deprecations to reduce technical debt and prepare for some new features and subsystems! The following deprecated APIs and commands will be removed on 2023-10-01:

17.219.1 Storm Commands

- `sudo`
- `splice.list`
- `splice.undo`

17.219.2 Storm Options

- `editformat=splices`

17.219.3 Cortex Telepath APIs

- `stat()`
- `addCronJob()`
- `delCronJob()`
- `updateCronJob()`
- `enableCronJob()`
- `disableCronJob()`
- `listCronJobs()`
- `editCronJob()`
- `setStormCmd()`
- `delStormCmd()`
- `addNodeTag()`
- `delNodeTag()`
- `setNodeProp()`
- `delNodeProp()`
- `eval()`
- `watch()`
- `splices()`
- `splicesBack()`
- `spliceHistory()`
- `addFeedData(syn.splice, ...)`
- `addFeedData(syn.nodeedits, ...)`

17.219.4 Layer Telepath APIs

- `splices()`
- `splicesBack()`
- `truncate()`

17.219.5 Cmdr Commands

- `at`
- `cron`
- `trigger`

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

synapse, 569
synapse.axon, 871
synapse.cells, 886
synapse.cmds, 569
synapse.cmds.boss, 569
synapse.cmds.cortex, 570
synapse.cmds.hive, 571
synapse.common, 886
synapse.cortex, 895
synapse.cryotank, 922
synapse.daemon, 925
synapse.data, 571
synapse.datamodel, 926
synapse.exc, 930
synapse.glob, 938
synapse.lib, 572
synapse.lib.agenda, 611
synapse.lib.aha, 613
synapse.lib.ast, 617
synapse.lib.autodoc, 634
synapse.lib.base, 635
synapse.lib.boss, 641
synapse.lib.cache, 641
synapse.lib.cell, 642
synapse.lib.certdir, 662
synapse.lib.chop, 677
synapse.lib.cli, 679
synapse.lib.cmd, 682
synapse.lib.cmdr, 682
synapse.lib.config, 683
synapse.lib.const, 687
synapse.lib.coro, 687
synapse.lib.crypto, 572
synapse.lib.crypto.coin, 572
synapse.lib.crypto.ecc, 572
synapse.lib.crypto.passwd, 575
synapse.lib.crypto.rsa, 576
synapse.lib.crypto.tinfoil, 578
synapse.lib.datfile, 689
synapse.lib.dyndeps, 689
synapse.lib.encoding, 690
synapse.lib.gis, 691
synapse.lib.grammar, 692
synapse.lib.hashitem, 693
synapse.lib.hashset, 693
synapse.lib.health, 693
synapse.lib.hive, 694
synapse.lib.hiveauth, 697
synapse.lib.httpapi, 702
synapse.lib.ingest, 710
synapse.lib.interval, 710
synapse.lib.jsonstor, 711
synapse.lib.jupyter, 713
synapse.lib.layer, 718
synapse.lib.link, 728
synapse.lib.lmdbslab, 729
synapse.lib.modelrev, 737
synapse.lib.module, 738
synapse.lib.modules, 740
synapse.lib.msgpack, 740
synapse.lib.multislabseqn, 742
synapse.lib.nexus, 743
synapse.lib.node, 745
synapse.lib.oauth, 751
synapse.lib.output, 752
synapse.lib.parser, 752
synapse.lib.platforms, 579
synapse.lib.platforms.common, 579
synapse.lib.platforms.darwin, 580
synapse.lib.platforms.freebsd, 580
synapse.lib.platforms.linux, 580
synapse.lib.platforms.windows, 581
synapse.lib.queue, 754
synapse.lib.ratelimit, 755
synapse.lib.reflect, 755
synapse.lib.rstorm, 756
synapse.lib.schemas, 757
synapse.lib.scope, 757
synapse.lib.scrape, 759
synapse.lib.share, 764
synapse.lib.slaboffs, 764
synapse.lib.slabseqn, 764
synapse.lib.snap, 766

synapse.lib.spooled, 770
synapse.lib.storm, 771
synapse.lib.storm_format, 788
synapse.lib.stormctrl, 788
synapse.lib.stormhttp, 789
synapse.lib.stormlib, 581
synapse.lib.stormlib.aha, 581
synapse.lib.stormlib.auth, 582
synapse.lib.stormlib.backup, 585
synapse.lib.stormlib.basex, 585
synapse.lib.stormlib.cache, 585
synapse.lib.stormlib.cell, 586
synapse.lib.stormlib.compression, 586
synapse.lib.stormlib.cortex, 587
synapse.lib.stormlib.easyperm, 591
synapse.lib.stormlib.ethereum, 591
synapse.lib.stormlib.gen, 591
synapse.lib.stormlib.gis, 592
synapse.lib.stormlib.graph, 592
synapse.lib.stormlib.hashes, 592
synapse.lib.stormlib.hex, 593
synapse.lib.stormlib.imap, 593
synapse.lib.stormlib.infosec, 594
synapse.lib.stormlib.ipv6, 595
synapse.lib.stormlib.iters, 595
synapse.lib.stormlib.json, 596
synapse.lib.stormlib.log, 596
synapse.lib.stormlib.macro, 597
synapse.lib.stormlib.math, 597
synapse.lib.stormlib.mime, 598
synapse.lib.stormlib.model, 598
synapse.lib.stormlib.modelext, 600
synapse.lib.stormlib.notifications, 600
synapse.lib.stormlib.oauth, 601
synapse.lib.stormlib.pack, 601
synapse.lib.stormlib.project, 602
synapse.lib.stormlib.random, 604
synapse.lib.stormlib.scrape, 604
synapse.lib.stormlib.smtp, 605
synapse.lib.stormlib.spooled, 605
synapse.lib.stormlib.stats, 606
synapse.lib.stormlib.stix, 607
synapse.lib.stormlib.storm, 608
synapse.lib.stormlib.vault, 609
synapse.lib.stormlib.version, 610
synapse.lib.stormlib.xml, 610
synapse.lib.stormlib.yaml, 610
synapse.lib.stormsvc, 790
synapse.lib.stormtypes, 790
synapse.lib.stormwhois, 809
synapse.lib.structlog, 809
synapse.lib.task, 809
synapse.lib.thishost, 810
synapse.lib.thisplat, 810
synapse.lib.threads, 810
synapse.lib.time, 811
synapse.lib.trigger, 812
synapse.lib.types, 813
synapse.lib.urlhelp, 820
synapse.lib.version, 821
synapse.lib.view, 823
synapse.lookup, 826
synapse.lookup.cvss, 826
synapse.lookup.iana, 826
synapse.lookup.iso3166, 826
synapse.lookup.macho, 827
synapse.lookup.pe, 827
synapse.lookup.phonenum, 827
synapse.lookup.timezones, 827
synapse.mindmeld, 939
synapse.models, 827
synapse.models.auth, 828
synapse.models.base, 828
synapse.models.belief, 828
synapse.models.biz, 829
synapse.models.crypto, 829
synapse.models.dns, 829
synapse.models.economic, 829
synapse.models.files, 829
synapse.models.geopol, 830
synapse.models.geospace, 830
synapse.models.gov, 827
synapse.models.gov.cn, 827
synapse.models.gov.intl, 828
synapse.models.gov.us, 828
synapse.models.inet, 831
synapse.models.infotech, 833
synapse.models.language, 834
synapse.models.material, 835
synapse.models.media, 835
synapse.models.orgs, 835
synapse.models.person, 835
synapse.models.proj, 835
synapse.models.risk, 836
synapse.models.science, 836
synapse.models.syn, 836
synapse.models.telco, 837
synapse.models.transport, 838
synapse.servers, 838
synapse.servers.aha, 838
synapse.servers.axon, 838
synapse.servers.cell, 838
synapse.servers.cortex, 838
synapse.servers.cryotank, 838
synapse.servers.jsonstor, 838
synapse.servers.stemcell, 838
synapse.telepath, 939
synapse.tests, 838

synapse.tests.nopmod, 838
synapse.tests.utils, 839
synapse.tools, 857
synapse.tools.aha, 857
synapse.tools.aha.easycert, 857
synapse.tools.aha.enroll, 857
synapse.tools.aha.list, 857
synapse.tools.aha.provision, 857
synapse.tools.aha.provision.service, 857
synapse.tools.aha.provision.user, 857
synapse.tools.autodoc, 858
synapse.tools.axon2axon, 860
synapse.tools.backup, 860
synapse.tools.cellauth, 861
synapse.tools.cmdr, 861
synapse.tools.cryo, 857
synapse.tools.cryo.cat, 857
synapse.tools.cryo.list, 858
synapse.tools.csvtool, 861
synapse.tools.docker, 858
synapse.tools.docker.validate, 858
synapse.tools.easycert, 861
synapse.tools.feed, 861
synapse.tools.genpkg, 862
synapse.tools.guid, 862
synapse.tools.healthcheck, 863
synapse.tools.hive, 858
synapse.tools.hive.load, 858
synapse.tools.hive.save, 858
synapse.tools.json2mpk, 863
synapse.tools.livebackup, 863
synapse.tools.modrole, 863
synapse.tools.moduser, 863
synapse.tools.promote, 863
synapse.tools.pullfile, 863
synapse.tools.pushfile, 864
synapse.tools.reload, 864
synapse.tools.rstorm, 864
synapse.tools.storm, 864
synapse.utils, 867
synapse.utils.getrefs, 871
synapse.utils.stormcov, 867
synapse.utils.stormcov.plugin, 868

INDEX

A

abrvToByts() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
abrvToName() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
AbsPropCond (*class in synapse.lib.ast*), 617
ActiveV1 (*class in synapse.lib.httpapi*), 702
add() (*synapse.lib.agenda.Agenda method*), 611
add() (*synapse.lib.cache.TagGlobs method*), 642
add() (*synapse.lib.hive.Hive method*), 694
add() (*synapse.lib.hive.Node method*), 696
add() (*synapse.lib.hive.TeleHive method*), 697
add() (*synapse.lib.lmdbslab.Hist method*), 729
add() (*synapse.lib.lmdbslab.MultiQueue method*), 731
add() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
add() (*synapse.lib.scope.Scope method*), 758
add() (*synapse.lib.slabseqn.SlabSeqn method*), 764
add() (*synapse.lib.spooled.Set method*), 770
add() (*synapse.lib.stormlib.stix.StixBundle method*), 608
add() (*synapse.tests.utils.TstEnv method*), 856
add_argument() (*synapse.lib.storm.Parser method*), 781
addActiveCoro() (*synapse.lib.cell.Cell method*), 643
addAhaPool() (*synapse.lib.aha.AhaApi method*), 613
addAhaPool() (*synapse.lib.aha.AhaCell method*), 614
addAhaPoolSvc() (*synapse.lib.aha.AhaApi method*), 613
addAhaPoolSvc() (*synapse.lib.aha.AhaCell method*), 614
addAhaSvc() (*synapse.lib.aha.AhaApi method*), 613
addAhaSvc() (*synapse.lib.aha.AhaCell method*), 614
addAhaSvcProv() (*synapse.lib.aha.AhaApi method*), 613
addAhaSvcProv() (*synapse.lib.aha.AhaCell method*), 614
addAhaUrl() (*in module synapse.telepath*), 941
addAhaUserEnroll() (*synapse.lib.aha.AhaApi method*), 613
addAhaUserEnroll() (*synapse.lib.aha.AhaCell method*), 614
addAndSync() (*synapse.lib.hive.HiveApi method*), 695
addAuthGate() (*synapse.lib.hiveauth.Auth method*), 698
addAuthRole() (*synapse.lib.cell.CellApi method*), 655
addAuthRule() (*synapse.lib.cell.CellApi method*), 655
addBaseType() (*synapse.datamodel.Model method*), 927
addCertPath() (*in module synapse.lib.certdir*), 677
addCertPath() (*synapse.lib.certdir.CertDir method*), 662
addCmd() (*synapse.tests.utils.CmdGenerator method*), 839
addCmdClass() (*synapse.lib.cli.Cli method*), 679
addCoreQueue() (*synapse.cortex.Cortex method*), 901
addCreatorDeleterRoles() (*synapse.tests.utils.SynTest method*), 841
addCronEdits() (*synapse.cortex.Cortex method*), 901
addCronJob() (*synapse.cortex.Cortex method*), 901
addDataModels() (*synapse.datamodel.Model method*), 927
addDmon() (*synapse.lib.storm.DmonManager method*), 774
addEdge() (*synapse.datamodel.Model method*), 927
addEdge() (*synapse.lib.node.Node method*), 745
addEdge() (*synapse.lib.snap.ProtoNode method*), 766
addExcInfo() (*synapse.lib.ast.AstNode method*), 618
addExtModel() (*synapse.cortex.Cortex method*), 902
addExtModel() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
addFeedData() (*in module synapse.tools.feed*), 861
addFeedData() (*synapse.cortex.CoreApi method*), 896
addFeedData() (*synapse.cortex.Cortex method*), 902
addFeedData() (*synapse.lib.jupyter.CmdrCore method*), 713
addFeedData() (*synapse.lib.snap.Snap method*), 767
addFeedNodes() (*synapse.lib.snap.Snap method*), 767
addForm() (*synapse.cortex.CoreApi method*), 896
addForm() (*synapse.cortex.Cortex method*), 902
addForm() (*synapse.datamodel.Model method*), 927
addForm() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
addFormat() (*in module synapse.lib.encoding*), 690
addFormProp() (*synapse.cortex.CoreApi method*), 896

addFormProp() (*synapse.cortex.Cortex method*), 902
addFormProp() (*synapse.datamodel.Model method*), 928
addFormProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
addFromPath() (*synapse.lib.ast.EditNodeAdd method*), 620
addHead() (*synapse.lib.autodoc.RstHelp method*), 634
addHealthFunc() (*synapse.lib.cell.Cell method*), 643
addHttpApi() (*synapse.lib.cell.Cell method*), 643
addHttpApi() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
addHttpExtApi() (*synapse.cortex.Cortex method*), 902
addHttpSess() (*synapse.lib.cell.Cell method*), 643
addHttpsPort() (*synapse.lib.cell.Cell method*), 643
addIface() (*synapse.datamodel.Model method*), 928
addInput() (*synapse.lib.storm.Runtime method*), 782
addKid() (*synapse.lib.ast.AstNode method*), 618
addLayer() (*synapse.cortex.Cortex method*), 902
addLayer() (*synapse.lib.view.View method*), 823
addLayrPull() (*synapse.cortex.Cortex method*), 902
addLayrPush() (*synapse.cortex.Cortex method*), 902
addLibFuncs() (*synapse.lib.stormlib.auth.LibUser method*), 583
addLibFuncs() (*synapse.lib.stormtypes.Lib method*), 792
addLibFuncs() (*synapse.lib.stormtypes.LibJsonStor method*), 795
addLibFuncs() (*synapse.tests.utils.LibTst method*), 840
addLines() (*synapse.lib.autodoc.RstHelp method*), 634
addNode() (*synapse.cortex.CoreApi method*), 896
addNode() (*synapse.cortex.Cortex method*), 902
addNode() (*synapse.lib.snap.Snap method*), 767
addNode() (*synapse.lib.snap.SnapEditor method*), 770
addNode() (*synapse.lib.stormtypes.View method*), 807
addNode() (*synapse.lib.view.View method*), 823
addNodeEdits() (*synapse.lib.view.View method*), 823
addNodes() (*synapse.cortex.CoreApi method*), 896
addNodes() (*synapse.cortex.Cortex method*), 903
addNodes() (*synapse.lib.snap.Snap method*), 767
addNodeTag() (*synapse.cortex.Cortex method*), 902
addOAuthProvider() (*synapse.lib.oauth.OAuthMixin method*), 751
addQueue() (*synapse.lib.jsonstor.JsonStorApi method*), 711
addQueue() (*synapse.lib.jsonstor.JsonStorCell method*), 712
Addr (*class in synapse.models.inet*), 831
addReloadableSystem() (*synapse.lib.cell.Cell method*), 643
addResizeCallback() (*synapse.lib.lmdbslab.Slab method*), 732
addRole() (*synapse.lib.cell.Cell method*), 644
addRole() (*synapse.lib.cell.CellApi method*), 655
addRole() (*synapse.lib.hiveauth.Auth method*), 698
addRoleRule() (*synapse.lib.cell.Cell method*), 644
addRoleRule() (*synapse.lib.cell.CellApi method*), 655
addRule() (*synapse.lib.hiveauth.HiveRuler method*), 700
addRuntLift() (*synapse.cortex.Cortex method*), 903
addRuntPropDel() (*synapse.cortex.Cortex method*), 903
addRuntPropSet() (*synapse.cortex.Cortex method*), 903
addSignalHandlers() (*synapse.lib.base.Base method*), 635
addSignalHandlers() (*synapse.lib.cell.Cell method*), 644
addSignalHandlers() (*synapse.lib.cli.Cli method*), 679
addStormCmd() (*synapse.cortex.Cortex method*), 903
addStormDmon() (*synapse.cortex.CoreApi method*), 896
addStormDmon() (*synapse.cortex.Cortex method*), 903
addStormGraph() (*synapse.cortex.Cortex method*), 903
addStormLib() (*synapse.cortex.Cortex method*), 903
addStormLib() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
addStormMacro() (*synapse.cortex.Cortex method*), 903
addStormPkg() (*synapse.cortex.CoreApi method*), 896
addStormPkg() (*synapse.cortex.Cortex method*), 903
addStormRuntime() (*synapse.lib.snap.Snap method*), 768
addStormSvc() (*synapse.cortex.Cortex method*), 903
addStormType() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
addSvcToAha() (*synapse.tests.utils.SynTest method*), 841
addSvcToCore() (*synapse.tests.utils.SynTest method*), 842
addTag() (*synapse.lib.node.Node method*), 745
addTag() (*synapse.lib.snap.ProtoNode method*), 766
addTagProp() (*synapse.cortex.CoreApi method*), 896
addTagProp() (*synapse.cortex.Cortex method*), 904
addTagProp() (*synapse.datamodel.Model method*), 928
addTagProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
addTestBadReload() (*synapse.tests.utils.ReloadCell method*), 840
addTestRecords() (*synapse.tests.utils.TestModule method*), 854
addTestReload() (*synapse.tests.utils.ReloadCell method*), 840
addTrigger() (*synapse.lib.view.View method*), 823
addTrigQueue() (*synapse.lib.view.View method*), 823
addType() (*synapse.datamodel.Model method*), 928
addUnivProp() (*synapse.cortex.CoreApi method*), 896
addUnivProp() (*synapse.cortex.Cortex method*), 904
addUnivProp() (*synapse.datamodel.Model method*),

928
addUnivProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
addUser() (*synapse.lib.cell.Cell method*), 644
addUser() (*synapse.lib.cell.CellApi method*), 656
addUser() (*synapse.lib.hiveauth.Auth method*), 698
addUserApiKey() (*synapse.lib.cell.Cell method*), 644
addUserNotif() (*synapse.cortex.CoreApi method*), 896
addUserNotif() (*synapse.cortex.Cortex method*), 904
addUserNotif() (*synapse.lib.jsonstor.JsonStorApi method*), 711
addUserNotif() (*synapse.lib.jsonstor.JsonStorCell method*), 712
addUserRole() (*synapse.lib.cell.Cell method*), 644
addUserRole() (*synapse.lib.cell.CellApi method*), 656
addUserRule() (*synapse.lib.cell.Cell method*), 644
addUserRule() (*synapse.lib.cell.CellApi method*), 656
addVault() (*synapse.cortex.Cortex method*), 904
addView() (*synapse.cortex.Cortex method*), 904
addWebSock() (*synapse.lib.httpapi.Sess method*), 708
addWriteHold() (*synapse.lib.nexus.NexsRoot method*), 743
adminapi() (*in module synapse.lib.cell*), 662
agen() (*in module synapse.common*), 886
agen() (*in module synapse.lib.coro*), 687
Agenda (*class in synapse.lib.agenda*), 611
agenlen() (*synapse.tests.utils.SynTest method*), 842
agenraises() (*synapse.tests.utils.SynTest method*), 842
aget() (*synapse.lib.cache.FixedCache method*), 641
AhaApi (*class in synapse.lib.aha*), 613
AhaCell (*class in synapse.lib.aha*), 614
AhaLib (*class in synapse.lib.stormlib.aha*), 581
AhaPool (*class in synapse.lib.stormlib.aha*), 581
AhaPoolLib (*class in synapse.lib.stormlib.aha*), 581
AhaProvisionServiceV1 (*class in synapse.lib.aha*), 616
AhaServicesV1 (*class in synapse.lib.aha*), 617
aiter() (*synapse.lib.slabseqn.SlabSeqn method*), 764
alias() (*in module synapse.telepath*), 941
alist() (*in module synapse.tests.utils*), 856
alist() (*synapse.lib.parser.CmdStringer method*), 753
allow() (*synapse.lib.hiveauth.HiveUser method*), 700
allowed() (*in module synapse.lib.stormtypes*), 807
allowed() (*synapse.lib.cell.CellApi method*), 656
allowed() (*synapse.lib.hiveauth.HiveRole method*), 700
allowed() (*synapse.lib.hiveauth.HiveUser method*), 700
allowed() (*synapse.lib.httpapi.HandlerBase method*), 705
allowed() (*synapse.lib.storm.Runtime method*), 782
allowedEasyPerm() (*in module synapse.lib.stormtypes*), 808
allowedEasyPerm() (*synapse.lib.storm.Runtime method*), 782
allowedReason() (*synapse.lib.storm.Runtime method*), 782
allows() (*synapse.lib.ratelimit.RateLimit method*), 755
allslabs (*synapse.lib.lmdbslab.Slab attribute*), 732
AndCond (*class in synapse.lib.ast*), 617
anit() (*synapse.lib.base.Base class method*), 636
applyNodeEdit() (*synapse.lib.snap.Snap method*), 768
applyNodeEdits() (*synapse.lib.snap.Snap method*), 768
ApptRec (*class in synapse.lib.agenda*), 612
AQueue (*class in synapse.lib.queue*), 754
Area (*class in synapse.models.geospace*), 830
ArgvQuery (*class in synapse.lib.ast*), 617
Array (*class in synapse.lib.types*), 813
ArrayCond (*class in synapse.lib.ast*), 618
asDict() (*synapse.lib.config.Config method*), 683
aspin() (*in module synapse.common*), 886
asroot (*synapse.lib.storm.Cmd attribute*), 772
assetdir (*synapse.tests.utils.StormPkgTest attribute*), 840
AstConverter (*class in synapse.lib.parser*), 752
AstInfo (*class in synapse.lib.parser*), 753
AstNode (*class in synapse.lib.ast*), 618
AsyncGenr (*class in synapse.daemon*), 925
asyncraises() (*synapse.tests.utils.SynTest method*), 842
AsyncStreamEvent (*class in synapse.tests.utils*), 839
Auth (*class in synapse.lib.hiveauth*), 697
AuthAddRoleV1 (*class in synapse.lib.httpapi*), 702
AuthAddUserV1 (*class in synapse.lib.httpapi*), 702
AuthDelRoleV1 (*class in synapse.lib.httpapi*), 702
AuthDeny, 930
authenticated() (*synapse.lib.httpapi.HandlerBase method*), 705
AuthGate (*class in synapse.lib.hiveauth*), 700
AuthGrantV1 (*class in synapse.lib.httpapi*), 702
AuthModule (*class in synapse.models.auth*), 828
AuthRevokeV1 (*class in synapse.lib.httpapi*), 702
AuthRolesV1 (*class in synapse.lib.httpapi*), 702
AuthRoleV1 (*class in synapse.lib.httpapi*), 702
AuthUserPasswdV1 (*class in synapse.lib.httpapi*), 703
AuthUsersV1 (*class in synapse.lib.httpapi*), 703
AuthUserV1 (*class in synapse.lib.httpapi*), 703
Aware (*class in synapse.telepath*), 939
Axon (*class in synapse.axon*), 871
AxonApi (*class in synapse.axon*), 878
AxonFileHandler (*class in synapse.axon*), 884
AxonHandlerMixin (*class in synapse.axon*), 884
AxonHttpBySha256InvalidV1 (*class in synapse.axon*), 884
AxonHttpBySha256V1 (*class in synapse.axon*), 885
AxonHttpDelV1 (*class in synapse.axon*), 885
AxonHttpHasV1 (*class in synapse.axon*), 885
AxonHttpUploadV1 (*class in synapse.axon*), 885

B

BackgroundCmd (*class in synapse.lib.storm*), 771
backup() (*in module synapse.tools.backup*), 860
backup_lmdb() (*in module synapse.tools.backup*), 860
BACKUP_SPAWN_TIMEOUT (*synapse.lib.cell.Cell attribute*), 643
BackupAlreadyRunning, 930
BackupLib (*class in synapse.lib.stormlib.backup*), 585
BadArg, 930
BadCast, 930
BadCertBytes, 930
BadCertHost, 930
BadCertVerify, 930
BadCmdName, 930
BadCmprType, 930
BadCmprValu, 930
BadConfValu, 930
BadCoreStore, 930
BadCtorType, 930
BadDataValu, 930
BadEccExchange, 931
BadFileExt, 931
BadFormDef, 931
BadHivePath, 931
BadIndxValu, 931
BadJsonText, 931
BadLiftValu, 931
BadMesgFormat, 931
BadMesgVers, 931
BadOperArg, 931
BadOptValu, 931
BadPkgDef, 931
BadPropDef, 931
BadState, 931
BadStorageVersion, 931
BadSyntax, 931
BadTag, 931
BadTime, 932
BadTypeDef, 932
BadTypeValu, 932
BadUrl, 932
BadVersion, 932
Base (*class in synapse.lib.base*), 635
base_undefined_types (*synapse.lib.stormtypes.StormTypesRegistry attribute*), 805
BaseModule (*class in synapse.models.base*), 828
BaseRef (*class in synapse.lib.base*), 639
BaseXLib (*class in synapse.lib.stormlib.basex*), 585
BatchCmd (*class in synapse.lib.storm*), 771
bbox() (*in module synapse.lib.gis*), 691
bch_check() (*in module synapse.lib.crypto.coin*), 572
beep() (*synapse.tests.utils.LibTst method*), 840
behold() (*synapse.lib.cell.Cell method*), 644
behold() (*synapse.lib.cell.CellApi method*), 656
beholder() (*synapse.lib.cell.Cell method*), 644
BeholdSockV1 (*class in synapse.lib.httpapi*), 703
BeliefModule (*class in synapse.models.belief*), 828
BizModule (*class in synapse.models.biz*), 829
Bool (*class in synapse.lib.ast*), 618
Bool (*class in synapse.lib.stormtypes*), 790
Bool (*class in synapse.lib.types*), 813
bool() (*synapse.lib.stormtypes.Prim method*), 802
Boss (*class in synapse.lib.boss*), 641
BreakOper (*class in synapse.lib.ast*), 619
bruteVersionStr() (*synapse.models.infotech.ItModule method*), 833
btc_base58_check() (*in module synapse.lib.crypto.coin*), 572
btc_bech32_check() (*in module synapse.lib.crypto.coin*), 572
buid() (*in module synapse.common*), 886
buidcachesize (*synapse.lib.snap.Snap attribute*), 768
buidsByDups() (*synapse.lib.layer.IndxBy method*), 719
buidsByPref() (*synapse.lib.layer.IndxBy method*), 719
buidsByRange() (*synapse.lib.layer.IndxBy method*), 719
buidsByRangeBack() (*synapse.lib.layer.IndxBy method*), 719
buildfilter() (*synapse.lib.ast.N1Walk method*), 626
buildgenr() (*synapse.lib.ast.FormPivot method*), 623
buildgenr() (*synapse.lib.ast.PropPivot method*), 628
bump() (*synapse.lib.lmdbslab.Scan method*), 731
bump() (*synapse.lib.storm.StormDmon method*), 785
bumpStormDmon() (*synapse.cortex.CoreApi method*), 896
bumpStormDmon() (*synapse.cortex.Cortex method*), 904
bundle() (*synapse.lib.stormlib.stix.LibStixExport method*), 607
byterange (*synapse.axon.Axon attribute*), 871
Bytes (*class in synapse.lib.stormtypes*), 790
bytsToAbrv() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
Bzip2Lib (*class in synapse.lib.stormlib.compression*), 586

C

cachedel() (*synapse.lib.stormtypes.LibJsonStor method*), 795
cacheget() (*synapse.lib.stormtypes.LibJsonStor method*), 795
cacheget() (*synapse.lib.stormtypes.NodeData method*), 800
cacheset() (*synapse.lib.stormtypes.LibJsonStor method*), 795
cacheset() (*synapse.lib.stormtypes.NodeData method*), 800

calculate() (*synapse.lib.stormlib.infosec.CvssLib method*), 594
calculateFromProps() (*synapse.lib.stormlib.infosec.CvssLib method*), 594
call() (*synapse.telepath.Proxy method*), 940
CallArgs (*class in synapse.lib.ast*), 619
callfunc() (*synapse.lib.ast.Function method*), 624
CallKwarg (*class in synapse.lib.ast*), 619
CallKwargs (*class in synapse.lib.ast*), 619
callStorm() (*synapse.cortex.CoreApi method*), 896
callStorm() (*synapse.cortex.Cortex method*), 904
callStorm() (*synapse.lib.view.View method*), 823
callStormIface() (*synapse.lib.view.View method*), 823
cancel() (*synapse.lib.storm.Runtime method*), 782
CantDelCmd, 932
CantDelForm, 932
CantDelNode, 932
CantDelProp, 932
CantDelType, 932
CantDelUniv, 932
CantDelView, 932
CantMergeView, 932
CantRevLayer, 932
capturelmdbs() (*in module synapse.tools.backup*), 860
cardano_byron_check() (*in module synapse.lib.crypto.coin*), 572
cardano_shelly_check() (*in module synapse.lib.crypto.coin*), 572
carve() (*synapse.lib.lmdbslab.Hist method*), 729
CaseEntry (*class in synapse.lib.ast*), 619
CatchBlock (*class in synapse.lib.ast*), 619
catches() (*synapse.lib.ast.CatchBlock method*), 619
Cell (*class in synapse.lib.cell*), 642
CellApi (*class in synapse.lib.cell*), 655
cellapi (*synapse.axon.Axon attribute*), 871
cellapi (*synapse.cortex.Cortex attribute*), 904
cellapi (*synapse.cryotank.CryoCell attribute*), 922
cellapi (*synapse.lib.aha.AhaCell attribute*), 614
cellapi (*synapse.lib.cell.Cell attribute*), 644
cellapi (*synapse.lib.jsonstor.JsonStorCell attribute*), 712
CellLib (*class in synapse.lib.stormlib.cell*), 586
CertDir (*class in synapse.lib.certdir*), 662
ChangeDist (*class in synapse.lib.nexus*), 743
check_origin() (*synapse.lib.httpapi.HandlerBase method*), 705
checkCosign() (*in module synapse.tools.docker.validate*), 858
checkCosignSignature() (*in module synapse.tools.docker.validate*), 858
checkCRL() (*in module synapse.tools.docker.validate*), 858
checkFreeSpace() (*synapse.lib.cell.Cell method*), 644
checkNode() (*synapse.tests.utils.SynTest method*), 842
checkNodes() (*synapse.tests.utils.SynTest method*), 842
checkShadowV2() (*in module synapse.lib.crypto.passwd*), 575
checkUserApiKey() (*synapse.lib.cell.Cell method*), 644
checkUserApiKey() (*synapse.lib.cell.CellApi method*), 656
checkUserLimit() (*synapse.lib.hiveauth.Auth method*), 699
chop_float() (*in module synapse.lib.grammar*), 692
chop_imei() (*in module synapse.models.telco*), 837
chopCpe22() (*in module synapse.models.infotech*), 834
chopurl() (*in module synapse.lib.urlhelp*), 820
chopurl() (*in module synapse.telepath*), 942
chunks() (*in module synapse.common*), 887
Cidr4 (*class in synapse.models.inet*), 831
Cidr6 (*class in synapse.models.inet*), 831
clear() (*synapse.lib.cache.FixedCache method*), 641
clear() (*synapse.tests.utils.TstOutPut method*), 856
clearAhaSvcProvs() (*synapse.lib.aha.AhaApi method*), 613
clearAhaSvcProvs() (*synapse.lib.aha.AhaCell method*), 614
clearAhaUserEnrolls() (*synapse.lib.aha.AhaApi method*), 613
clearAhaUserEnrolls() (*synapse.lib.aha.AhaCell method*), 614
clearAuthCache() (*synapse.lib.hiveauth.HiveRole method*), 700
clearAuthCache() (*synapse.lib.hiveauth.HiveUser method*), 701
clearCache() (*synapse.lib.snap.Snap method*), 768
clearCachedNode() (*synapse.lib.snap.Snap method*), 768
clearOAuthAccessToken() (*synapse.lib.oauth.OAuthMixin method*), 751
clearRunningStatus() (*synapse.lib.agenda.Agenda method*), 611
Cli (*class in synapse.lib.cli*), 679
Client (*class in synapse.telepath*), 939
ClientV2 (*class in synapse.telepath*), 939
CliFini, 932
clone() (*in module synapse.lib.scope*), 758
clone() (*synapse.lib.layer.Layer method*), 720
clone() (*synapse.lib.node.Path method*), 747
clone() (*synapse.lib.types.Type method*), 818
cloneLayer() (*synapse.cortex.CoreApi method*), 896
cloneLayer() (*synapse.cortex.Cortex method*), 904
close() (*synapse.lib.queue.Queue method*), 754
close() (*synapse.lib.stormtypes.Pipe method*), 802
closeLogFd() (*synapse.cmds.cortex.Log method*), 570
Cmd (*class in synapse.lib.cli*), 680

Cmd (*class in synapse.lib.storm*), 771
CmdGenerator (*class in synapse.tests.utils*), 839
CmdHelp (*class in synapse.lib.cli*), 681
CmdLocals (*class in synapse.lib.cli*), 681
CmdOper (*class in synapse.lib.ast*), 619
CmdOpts (*class in synapse.lib.stormtypes*), 790
CmdQuit (*class in synapse.lib.cli*), 681
cmdrargs() (*synapse.lib.parser.AstConverter method*), 752
cmdrargs() (*synapse.lib.parser.Parser method*), 754
CmdrCore (*class in synapse.lib.jupyter*), 713
cmdstring() (*synapse.lib.parser.CmdStringer method*), 753
CmdStringer (*class in synapse.lib.parser*), 753
cmpDelPathObjProp() (*synapse.lib.jsonstor.JsonStor method*), 711
cmpDelPathObjProp() (*synapse.lib.jsonstor.JsonStorApi method*), 711
cmpDelPathObjProp() (*synapse.lib.jsonstor.JsonStorCell method*), 712
cmplgenr() (*in module synapse.tools.storm*), 867
Cmpr (*class in synapse.lib.ast*), 619
cmpr() (*synapse.lib.types.Type method*), 818
cmprkey_buid() (*in module synapse.cortex*), 922
cmprkey_indx() (*in module synapse.cortex*), 922
codereason() (*synapse.lib.stormhttp.LibHttp method*), 789
COMMIT (*synapse.lib.cell.Cell attribute*), 643
COMMIT_PERIOD (*synapse.lib.lmdbslab.Slab attribute*), 732
Comp (*class in synapse.lib.types*), 813
compileJsSchema() (*in module synapse.lib.stormlib.json*), 596
compute() (*synapse.lib.ast.ArgvQuery method*), 617
compute() (*synapse.lib.ast.CallArgs method*), 619
compute() (*synapse.lib.ast.Const method*), 619
compute() (*synapse.lib.ast.DollarExpr method*), 620
compute() (*synapse.lib.ast.EmbedQuery method*), 621
compute() (*synapse.lib.ast.ExprAndNode method*), 622
compute() (*synapse.lib.ast.ExprDict method*), 622
compute() (*synapse.lib.ast.ExprList method*), 622
compute() (*synapse.lib.ast.ExprNode method*), 622
compute() (*synapse.lib.ast.ExprOrNode method*), 622
compute() (*synapse.lib.ast.FormatString method*), 623
compute() (*synapse.lib.ast.FormName method*), 623
compute() (*synapse.lib.ast.FormTagProp method*), 623
compute() (*synapse.lib.ast.FuncArgs method*), 623
compute() (*synapse.lib.ast.FuncCall method*), 623
compute() (*synapse.lib.ast.List method*), 626
compute() (*synapse.lib.ast.PropName method*), 627
compute() (*synapse.lib.ast.PropValue method*), 628
compute() (*synapse.lib.ast.SubQuery method*), 630
compute() (*synapse.lib.ast.TagMatch method*), 631
compute() (*synapse.lib.ast.TagName method*), 631
compute() (*synapse.lib.ast.TagProp method*), 631
compute() (*synapse.lib.ast.TagPropValue method*), 631
compute() (*synapse.lib.ast.TagValue method*), 631
compute() (*synapse.lib.ast.UnaryExprNode method*), 632
compute() (*synapse.lib.ast.UnivProp method*), 632
compute() (*synapse.lib.ast.Value method*), 632
compute() (*synapse.lib.ast.VarDeref method*), 632
compute() (*synapse.lib.ast.VarValue method*), 633
compute_array() (*synapse.lib.ast.SubQuery method*), 630
compute_etag() (*synapse.lib.httpapi.ExtApiHandler method*), 703
computeTagArray() (*synapse.lib.ast.TagName method*), 631
concat() (*synapse.lib.stormtypes.LibStr method*), 798
Cond (*class in synapse.lib.ast*), 619
confbase (*synapse.cortex.Cortex attribute*), 904
confbase (*synapse.lib.aha.AhaCell attribute*), 614
confbase (*synapse.lib.cell.Cell attribute*), 645
confdefs (*synapse.axon.Axon attribute*), 871
confdefs (*synapse.cortex.Cortex attribute*), 906
confdefs (*synapse.lib.aha.AhaCell attribute*), 616
confdefs (*synapse.lib.cell.Cell attribute*), 647
confdefs (*synapse.lib.module.CoreModule attribute*), 738
Config (*class in synapse.lib.config*), 683
config() (*in module synapse.common*), 887
config() (*synapse.lib.stormlib.stix.LibStixExport method*), 607
config() (*synapse.lib.stormlib.stix.LibStixImport method*), 608
confirm() (*in module synapse.lib.stormtypes*), 808
confirm() (*synapse.lib.hiveauth.HiveUser method*), 701
confirm() (*synapse.lib.storm.Runtime method*), 782
confirm() (*synapse.lib.stormlib.project.Project method*), 602
confirmEasyPerm() (*in module synapse.lib.stormtypes*), 808
confirmEasyPerm() (*synapse.lib.storm.Runtime method*), 783
confirmPropDel() (*synapse.lib.storm.Runtime method*), 783
confirmPropSet() (*synapse.lib.storm.Runtime method*), 783
connect() (*in module synapse.lib.link*), 729
connect() (*synapse.lib.stormlib.imap.ImapLib method*), 593
Const (*class in synapse.lib.ast*), 619
contextScrape() (*in module synapse.lib.scrape*), 759
contextScrapeAsync() (*in module synapse.lib.scrape*), 760

ContinueOper (*class in synapse.lib.ast*), 619
copy() (*synapse.lib.scope.Scope method*), 758
copydb() (*synapse.lib.lmdbslab.Slab method*), 732
copyPathObj() (*synapse.lib.jsonstor.JsonStor method*), 711
copyPathObj() (*synapse.lib.jsonstor.JsonStorApi method*), 711
copyPathObj() (*synapse.lib.jsonstor.JsonStorCell method*), 712
copyPathObjs() (*synapse.lib.jsonstor.JsonStor method*), 711
copyPathObjs() (*synapse.lib.jsonstor.JsonStorApi method*), 712
copyPathObjs() (*synapse.lib.jsonstor.JsonStorCell method*), 712
copyslab() (*synapse.lib.lmdbslab.Slab method*), 733
CopyToCmd (*class in synapse.lib.storm*), 772
CoreApi (*class in synapse.cortex*), 895
coreDynCall() (*synapse.lib.storm.Runtime method*), 783
CoreInfoV1 (*class in synapse.lib.httpapi*), 703
CoreModule (*class in synapse.lib.module*), 738
coreQueueCull() (*synapse.cortex.Cortex method*), 906
coreQueueGet() (*synapse.cortex.Cortex method*), 906
coreQueueGets() (*synapse.cortex.Cortex method*), 906
coreQueuePop() (*synapse.cortex.Cortex method*), 906
coreQueuePuts() (*synapse.cortex.Cortex method*), 906
coreQueueSize() (*synapse.cortex.Cortex method*), 906
Cortex (*class in synapse.cortex*), 901
CortexAxonHttpBySha256InvalidV1 (*class in synapse.cortex*), 921
CortexAxonHttpBySha256V1 (*class in synapse.cortex*), 921
CortexAxonHttpDelV1 (*class in synapse.cortex*), 921
CortexAxonHttpHasV1 (*class in synapse.cortex*), 921
CortexAxonHttpUploadV1 (*class in synapse.cortex*), 921
CortexAxonMixin (*class in synapse.cortex*), 921
CortexHttpApi (*class in synapse.lib.stormlib.cortex*), 587
count() (*synapse.cortex.CoreApi method*), 896
count() (*synapse.cortex.Cortex method*), 906
count() (*synapse.lib.lmdbslab.Slab method*), 733
countByPref() (*synapse.lib.lmdbslab.Slab method*), 733
CountCmd (*class in synapse.lib.storm*), 773
coverage_init() (*in module synapse.utils.stormcov*), 867
Cpe22Str (*class in synapse.models.infotech*), 833
Cpe23Str (*class in synapse.models.infotech*), 833
cpesplit() (*in module synapse.models.infotech*), 834
Crl (*class in synapse.lib.certdir*), 676
CronJob (*class in synapse.lib.stormtypes*), 791
CryoApi (*class in synapse.cryotank*), 922
CryoCell (*class in synapse.cryotank*), 922
CryoTank (*class in synapse.cryotank*), 923
CryptoErr, 932
CryptoModule (*class in synapse.models.crypto*), 829
CryptSeq (*class in synapse.lib.crypto.tinfoil*), 578
csvrows() (*synapse.axon.Axon method*), 871
csvrows() (*synapse.axon.AxonApi method*), 878
csvrows() (*synapse.lib.stormtypes.LibAxon method*), 792
ctor() (*in module synapse.lib.scope*), 759
cull() (*synapse.lib.lmdbslab.MultiQueue method*), 731
cull() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
cull() (*synapse.lib.nexus.NexsRoot method*), 743
cull() (*synapse.lib.slabseqn.SlabSeqn method*), 764
cullNexsLog() (*synapse.lib.cell.Cell method*), 647
cullNexsLog() (*synapse.lib.cell.CellApi method*), 656
cullQueue() (*synapse.lib.jsonstor.JsonStorApi method*), 712
cullQueue() (*synapse.lib.jsonstor.JsonStorCell method*), 712
current() (*in module synapse.lib.task*), 809
current() (*in module synapse.lib.threads*), 810
cve_check() (*in module synapse.lib.scrape*), 760
CVSS2_calc() (*in module synapse.lib.stormlib.infosec*), 594
cvss2_normalize() (*in module synapse.lib.chop*), 677
CVSS2_round() (*in module synapse.lib.stormlib.infosec*), 594
CVSS3_0_calc() (*in module synapse.lib.stormlib.infosec*), 594
CVSS3_0_round() (*in module synapse.lib.stormlib.infosec*), 594
CVSS3_1_calc() (*in module synapse.lib.stormlib.infosec*), 594
CVSS3_1_round() (*in module synapse.lib.stormlib.infosec*), 594
cvss3x_normalize() (*in module synapse.lib.chop*), 677
CVSS_get_coefficients() (*in module synapse.lib.stormlib.infosec*), 594
cvss_normalize() (*in module synapse.lib.chop*), 677
cvss_validate() (*in module synapse.lib.chop*), 677
CvssLib (*class in synapse.lib.stormlib.infosec*), 594
CvssV2 (*class in synapse.models.risk*), 836
CvssV3 (*class in synapse.models.risk*), 836

D

Daemon (*class in synapse.daemon*), 925
daemonize() (*in module synapse.lib.platforms.common*), 579
daemonize() (*in module synapse.lib.platforms.windows*), 581
Data (*class in synapse.lib.types*), 813

data_received() (*synapse.axon.AxonHttpUploadV1 method*), 885
data_received() (*synapse.lib.httpapi.StreamHandler method*), 710
DataAlreadyExists, 932
DAY (*synapse.lib.agenda.TimeUnit attribute*), 612
day() (*in module synapse.lib.time*), 811
day() (*synapse.lib.stormtypes.LibTime method*), 798
DAYOFMONTH (*synapse.lib.agenda.TimeUnit attribute*), 612
dayofmonth() (*in module synapse.lib.time*), 811
dayofmonth() (*synapse.lib.stormtypes.LibTime method*), 798
DAYOFWEEK (*synapse.lib.agenda.TimeUnit attribute*), 612
dayofweek() (*in module synapse.lib.time*), 811
dayofweek() (*synapse.lib.stormtypes.LibTime method*), 798
dayofyear() (*in module synapse.lib.time*), 811
dayofyear() (*synapse.lib.stormtypes.LibTime method*), 798
dbexists() (*synapse.lib.lmdbslab.Slab method*), 733
DbOutOfSpace, 933
debase64() (*in module synapse.common*), 887
dec() (*synapse.lib.crypto.tinfoil.TinFoilHat method*), 578
DecFunc() (*synapse.lib.lmdbslab.HotCount static method*), 729
DecFunc() (*synapse.lib.lmdbslab.HotKeyVal static method*), 730
decode() (*in module synapse.lib.encoding*), 690
decode() (*synapse.lib.stormlib.basex.BaseXLib method*), 585
decode() (*synapse.lib.stormlib.hex.HexLib method*), 593
decodeIndx() (*synapse.lib.layer.StorType method*), 726
decodeIndx() (*synapse.lib.layer.StorTypeFloat method*), 726
decodeIndx() (*synapse.lib.layer.StorTypeFqdn method*), 726
decodeIndx() (*synapse.lib.layer.StorTypeGuid method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeHier method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeHugeNum method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeInt method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeIpv6 method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeIval method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeLatLon method*), 727
decodeIndx() (*synapse.lib.layer.StorTypeUtf8 method*), 728
decrypt() (*synapse.lib.crypto.tinfoil.CryptSeq method*), 578
deepcopy() (*in module synapse.lib.msgpack*), 740
DEFAULT_GROWSIZE (*synapse.lib.lmdbslab.Slab attribute*), 732
DEFAULT_MAPSIZE (*synapse.lib.lmdbslab.Slab attribute*), 732
deguidify() (*in module synapse.tests.utils*), 856
del_() (*synapse.axon.Axon method*), 871
del_() (*synapse.axon.AxonApi method*), 879
del_() (*synapse.lib.lmdbslab.GuidStor method*), 729
del_() (*synapse.lib.stormtypes.LibAxon method*), 792
delActiveCoro() (*synapse.lib.cell.Cell method*), 647
delAhaPool() (*synapse.lib.aha.AhaApi method*), 613
delAhaPool() (*synapse.lib.aha.AhaCell method*), 616
delAhaPoolSvc() (*synapse.lib.aha.AhaApi method*), 613
delAhaPoolSvc() (*synapse.lib.aha.AhaCell method*), 616
delAhaSvc() (*synapse.lib.aha.AhaApi method*), 613
delAhaSvc() (*synapse.lib.aha.AhaCell method*), 616
delAhaSvcProv() (*synapse.lib.aha.AhaApi method*), 613
delAhaSvcProv() (*synapse.lib.aha.AhaCell method*), 616
delAhaUrl() (*in module synapse.telepath*), 942
delAhaUserEnroll() (*synapse.lib.aha.AhaApi method*), 613
delAhaUserEnroll() (*synapse.lib.aha.AhaCell method*), 616
delAuthGate() (*synapse.lib.hiveauth.Auth method*), 699
delAuthRole() (*synapse.lib.cell.CellApi method*), 656
delAuthRule() (*synapse.lib.cell.CellApi method*), 656
delAuthUser() (*synapse.lib.cell.CellApi method*), 657
delBackup() (*synapse.lib.cell.Cell method*), 647
delBackup() (*synapse.lib.cell.CellApi method*), 657
delCertPath() (*in module synapse.lib.certdir*), 677
delCertPath() (*synapse.lib.certdir.CertDir method*), 662
delCoreQueue() (*synapse.cortex.Cortex method*), 906
delCronJob() (*synapse.cortex.Cortex method*), 906
dele() (*synapse.lib.lmdbslab.MultiQueue method*), 731
delEdge() (*synapse.lib.node.Node method*), 745
delEdge() (*synapse.lib.snap.ProtoNode method*), 766
delEdges() (*synapse.lib.storm.EdgesDelCmd method*), 775
delete() (*synapse.axon.AxonHttpBySha256InvalidV1 method*), 884
delete() (*synapse.axon.AxonHttpBySha256V1 method*), 885
delete() (*synapse.cryotank.CryoApi method*), 922
delete() (*synapse.cryotank.CryoCell method*), 923

delete() (*synapse.lib.agenda.Agenda method*), 612
delete() (*synapse.lib.hiveauth.AuthGate method*), 700
delete() (*synapse.lib.httpapi.ExtApiHandler method*), 703
delete() (*synapse.lib.layer.Layer method*), 720
delete() (*synapse.lib.lmdbslab.HotKeyVal method*), 730
delete() (*synapse.lib.lmdbslab.Slab method*), 733
delete() (*synapse.lib.node.Node method*), 745
delete() (*synapse.lib.slaboffs.SlabOffs method*), 764
delete() (*synapse.lib.stormlib.imap.ImapServer method*), 593
delete() (*synapse.lib.view.View method*), 823
delForm() (*synapse.cortex.CoreApi method*), 897
delForm() (*synapse.cortex.Cortex method*), 906
delForm() (*synapse.datamodel.Model method*), 928
delForm() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
delFormProp() (*synapse.cortex.CoreApi method*), 897
delFormProp() (*synapse.cortex.Cortex method*), 906
delFormProp() (*synapse.datamodel.Model method*), 928
delFormProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
delHttpApi() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
delHttpExtApi() (*synapse.cortex.Cortex method*), 906
delHttpSess() (*synapse.lib.cell.Cell method*), 647
delJsonObj() (*synapse.cortex.Cortex method*), 907
delJsonObjProp() (*synapse.cortex.Cortex method*), 907
delLayer() (*synapse.cortex.Cortex method*), 907
delLayrPull() (*synapse.cortex.Cortex method*), 907
delLayrPush() (*synapse.cortex.Cortex method*), 907
delMergeRequest() (*synapse.lib.stormtypes.View method*), 807
delMergeRequest() (*synapse.lib.view.View method*), 823
delMergeVote() (*synapse.lib.stormtypes.View method*), 807
delMergeVote() (*synapse.lib.view.View method*), 823
DelNodeCmd (*class in synapse.lib.storm*), 773
delNodeTag() (*synapse.cortex.Cortex method*), 907
delOAuthProvider() (*synapse.lib.oauth.OAuthMixin method*), 751
delPathObj() (*synapse.lib.jsonstor.JsonStor method*), 711
delPathObj() (*synapse.lib.jsonstor.JsonStorApi method*), 712
delPathObj() (*synapse.lib.jsonstor.JsonStorCell method*), 712
delPathObjProp() (*synapse.lib.jsonstor.JsonStor method*), 711
delPathObjProp() (*synapse.lib.jsonstor.JsonStorApi method*), 712
delPathObjProp() (*synapse.lib.jsonstor.JsonStorCell method*), 712
delProp() (*synapse.datamodel.Form method*), 926
delQueue() (*synapse.lib.jsonstor.JsonStorApi method*), 712
delQueue() (*synapse.lib.jsonstor.JsonStorCell method*), 713
delRole() (*synapse.lib.cell.Cell method*), 647
delRole() (*synapse.lib.cell.CellApi method*), 657
delRole() (*synapse.lib.hiveauth.Auth method*), 699
delRoleRule() (*synapse.lib.cell.Cell method*), 647
delRoleRule() (*synapse.lib.cell.CellApi method*), 657
delRule() (*synapse.lib.hiveauth.HiveRuler method*), 700
dels() (*synapse.axon.Axon method*), 871
dels() (*synapse.axon.AxonApi method*), 879
dels() (*synapse.lib.stormtypes.LibAxon method*), 792
delStormCmd() (*synapse.cortex.Cortex method*), 907
delStormDmon() (*synapse.cortex.CoreApi method*), 897
delStormDmon() (*synapse.cortex.Cortex method*), 907
delStormGraph() (*synapse.cortex.Cortex method*), 907
delStormLib() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
delStormMacro() (*synapse.cortex.Cortex method*), 907
delStormPkg() (*synapse.cortex.CoreApi method*), 897
delStormPkg() (*synapse.cortex.Cortex method*), 907
delStormPool() (*synapse.cortex.Cortex method*), 907
delStormSvc() (*synapse.cortex.Cortex method*), 907
delStormType() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
delta() (*in module synapse.lib.time*), 811
delTag() (*synapse.lib.node.Node method*), 745
delTagModel() (*synapse.cortex.Cortex method*), 907
delTagProp() (*synapse.cortex.CoreApi method*), 897
delTagProp() (*synapse.cortex.Cortex method*), 907
delTagProp() (*synapse.datamodel.Model method*), 928
delTagProp() (*synapse.lib.node.Node method*), 745
delTagProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
delTrigger() (*synapse.lib.view.View method*), 823
delTrigQueue() (*synapse.lib.view.View method*), 823
delType() (*synapse.datamodel.Model method*), 928
delUnivProp() (*synapse.cortex.CoreApi method*), 897
delUnivProp() (*synapse.cortex.Cortex method*), 907
delUnivProp() (*synapse.datamodel.Model method*), 928
delUnivProp() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
delUser() (*synapse.lib.cell.Cell method*), 647
delUser() (*synapse.lib.cell.CellApi method*), 657
delUser() (*synapse.lib.hiveauth.Auth method*), 699
delUserApiKey() (*synapse.lib.cell.Cell method*), 647
delUserNotif() (*synapse.cortex.CoreApi method*), 897

delUserNotif() (*synapse.cortex.Cortex method*), 907
delUserNotif() (*synapse.lib.jsonstor.JsonStorApi method*), 712
delUserNotif() (*synapse.lib.jsonstor.JsonStorCell method*), 713
delUserRole() (*synapse.lib.cell.Cell method*), 647
delUserRole() (*synapse.lib.cell.CellApi method*), 657
delUserRule() (*synapse.lib.cell.Cell method*), 647
delUserRule() (*synapse.lib.cell.CellApi method*), 657
delVault() (*synapse.cortex.Cortex method*), 907
delView() (*synapse.cortex.Cortex method*), 907
delWebSock() (*synapse.lib.httpapi.Sess method*), 708
delWriteHold() (*synapse.lib.nexus.NexsRoot method*), 743
deprdate() (*in module synapse.common*), 887
deprecated() (*in module synapse.common*), 887
DeprModule (*class in synapse.tests.utils*), 839
deref() (*synapse.lib.stormlib.auth.UserProfile method*), 584
deref() (*synapse.lib.stormlib.auth.UserVars method*), 584
deref() (*synapse.lib.stormlib.cortex.HttpHeaderDict method*), 590
deref() (*synapse.lib.stormlib.vault.VaultConfigs method*), 609
deref() (*synapse.lib.stormtypes.CmdOpts method*), 790
deref() (*synapse.lib.stormtypes.Dict method*), 791
deref() (*synapse.lib.stormtypes.Lib method*), 792
deref() (*synapse.lib.stormtypes.PathMeta method*), 802
deref() (*synapse.lib.stormtypes.PathVars method*), 802
deref() (*synapse.lib.stormtypes.Proxy method*), 803
deref() (*synapse.lib.stormtypes.Service method*), 804
deref() (*synapse.lib.stormtypes.StormType method*), 805
deref() (*synapse.lib.stormtypes.Trigger method*), 806
detach() (*synapse.lib.stormtypes.View method*), 807
detach() (*synapse.lib.view.View method*), 823
Dict (*class in synapse.lib.spooled*), 770
Dict (*class in synapse.lib.stormtypes*), 791
dict() (*synapse.lib.hive.Hive method*), 694
dict() (*synapse.lib.hive.Node method*), 696
dict() (*synapse.lib.lmdbslab.GuidStor method*), 729
DiffCmd (*class in synapse.lib.storm*), 773
digests() (*synapse.lib.hashset.HashSet method*), 693
digits() (*in module synapse.lib.chop*), 678
digits() (*in module synapse.models.telco*), 837
dir() (*synapse.lib.hive.Hive method*), 694
dir() (*synapse.lib.hive.Node method*), 696
disable() (*synapse.lib.agenda.Agenda method*), 612
disableCronJob() (*synapse.cortex.Cortex method*), 907
disableMigrationMode() (*synapse.cortex.CoreApi method*), 897
disableStormDmon() (*synapse.cortex.CoreApi method*), 897
disableStormDmon() (*synapse.cortex.Cortex method*), 908
discard() (*synapse.lib.spooled.Set method*), 770
Dist (*class in synapse.models.geospace*), 830
dist() (*synapse.lib.base.Base method*), 636
DivertCmd (*class in synapse.lib.storm*), 774
dmonloop() (*synapse.lib.storm.StormDmon method*), 785
DmonManager (*class in synapse.lib.storm*), 774
DmonSpawn, 933
dms2dec() (*in module synapse.lib.gis*), 691
DnsModule (*class in synapse.models.dns*), 829
DnsName (*class in synapse.models.dns*), 829
do_handshake() (*synapse.telepath.TeleSSLObject method*), 941
docConfdefs() (*in module synapse.tools.autodoc*), 858
DocHelp (*class in synapse.tools.autodoc*), 858
docModel() (*in module synapse.tools.autodoc*), 858
docStormpkg() (*in module synapse.tools.autodoc*), 858
docStormsvc() (*in module synapse.tools.autodoc*), 859
docStormTypes() (*in module synapse.lib.autodoc*), 634
docStormTypes() (*in module synapse.tools.autodoc*), 858
doECDHE() (*in module synapse.lib.crypto.ecc*), 574
DollarExpr (*class in synapse.lib.ast*), 620
download_refs() (*in module synapse.utils.getrefs*), 871
download_refs_handler() (*in module synapse.utils.getrefs*), 871
dropdb() (*synapse.lib.lmdbslab.Slab method*), 733
dump() (*synapse.lib.crypto.ecc.PriKey method*), 572
dump() (*synapse.lib.crypto.ecc.PubKey method*), 574
dump() (*synapse.lib.crypto.rsa.PubKey method*), 577
dumpfile() (*in module synapse.lib.msgpack*), 740
DupFileName, 933
DupFormName, 933
DupIden, 933
DupIndx, 933
DupName, 933
DupPropName, 933
DupRoleName, 933
DupStormSvc, 933
DupTagPropName, 933
DupUserName, 933
Duration (*class in synapse.lib.types*), 814
dynamic_source_filename() (*synapse.utils.stormcov.plugin.PivotTracer method*), 868
dynamic_source_filename() (*synapse.utils.stormcov.plugin.StormCtrlTracer method*), 868
dynamic_source_filename() (*synapse.utils.stormcov.plugin.StormPlugin

method), 869
dyncall() (*synapse.lib.cell.Cell method*), 647
dyncall() (*synapse.lib.cell.CellApi method*), 657
dyncall() (*synapse.lib.storm.Runtime method*), 783
dyncall() (*synapse.lib.stormtypes.Lib method*), 792
dyniter() (*synapse.lib.cell.Cell method*), 647
dyniter() (*synapse.lib.cell.CellApi method*), 657
dyniter() (*synapse.lib.storm.Runtime method*), 783
dyniter() (*synapse.lib.stormtypes.Lib method*), 792

E

eat() (*synapse.lib.nexus.NexsRoot method*), 743
eatfd() (*synapse.lib.hashset.HashSet method*), 693
ecol (*synapse.lib.parser.AstInfo attribute*), 753
EconModule (*class in synapse.models.economic*), 829
Edge (*class in synapse.datamodel*), 926
Edge (*class in synapse.lib.types*), 814
EdgeGlobs (*class in synapse.lib.cache*), 641
EdgesDelCmd (*class in synapse.lib.storm*), 775
Edit (*class in synapse.lib.ast*), 620
editCronJob() (*synapse.cortex.Cortex method*), 908
EditEdgeAdd (*class in synapse.lib.ast*), 620
EditEdgeDel (*class in synapse.lib.ast*), 620
editformat_enums (*synapse.cmds.cortex.StormCmd attribute*), 571
EditNodeAdd (*class in synapse.lib.ast*), 620
EditParens (*class in synapse.lib.ast*), 620
EditPropDel (*class in synapse.lib.ast*), 620
EditPropSet (*class in synapse.lib.ast*), 620
edits() (*synapse.lib.hive.HiveApi method*), 695
EditTagAdd (*class in synapse.lib.ast*), 620
EditTagDel (*class in synapse.lib.ast*), 620
EditTagPropDel (*class in synapse.lib.ast*), 620
EditTagPropSet (*class in synapse.lib.ast*), 621
EditUnivDel (*class in synapse.lib.ast*), 621
ehex() (*in module synapse.common*), 887
eip55() (*synapse.lib.stormlib.ethereum.EthereumLib method*), 591
eline (*synapse.lib.parser.AstInfo attribute*), 753
Email (*class in synapse.models.inet*), 831
EmbedQuery (*class in synapse.lib.ast*), 621
embedquery() (*synapse.lib.parser.AstConverter method*), 752
Emit (*class in synapse.lib.ast*), 621
emit() (*synapse.lib.storm.Runtime method*), 783
emitter() (*synapse.lib.storm.Runtime method*), 783
EmptyBlock (*class in synapse.lib.ast*), 621
en() (*in module synapse.lib.msgpack*), 740
en() (*synapse.lib.stormlib.compression.Bzip2Lib method*), 586
en() (*synapse.lib.stormlib.compression.GzipLib method*), 586
en() (*synapse.lib.stormlib.compression.ZlibLib method*), 587
en() (*synapse.lib.stormlib.pack.LibPack method*), 601
enable() (*synapse.lib.agenda.Agenda method*), 612
enableCronJob() (*synapse.cortex.Cortex method*), 908
enableMigrationMode() (*synapse.cortex.CoreApi method*), 897
enableStormDmon() (*synapse.cortex.CoreApi method*), 897
enableStormDmon() (*synapse.cortex.Cortex method*), 908
enbase64() (*in module synapse.common*), 887
enc() (*synapse.lib.crypto.tinfoil.TinFoilHat method*), 579
EncFunc() (*synapse.lib.lmdbslab.HotCount static method*), 729
EncFunc() (*synapse.lib.lmdbslab.HotKeyVal static method*), 730
encode() (*in module synapse.lib.encoding*), 690
encode() (*synapse.lib.stormlib.basex.BaseXLib method*), 585
encode() (*synapse.lib.stormlib.hex.HexLib method*), 593
encodeMsg() (*synapse.cmds.cortex.Log method*), 570
encrypt() (*synapse.lib.crypto.tinfoil.CryptSeq method*), 578
enNexsLog() (*synapse.lib.nexus.NexsRoot method*), 743
EnrollApi (*class in synapse.lib.aha*), 617
enter() (*in module synapse.lib.scope*), 759
enter() (*synapse.lib.scope.Scope method*), 758
enter_context() (*synapse.lib.base.Base method*), 636
enterMigrationMode() (*synapse.cortex.Cortex method*), 908
enum() (*synapse.lib.stormlib.iters.LibIters method*), 595
envbool() (*in module synapse.common*), 887
eoff (*synapse.lib.parser.AstInfo attribute*), 753
eq() (*synapse.tests.utils.SynTest method*), 842
eqish() (*synapse.tests.utils.SynTest method*), 842
eqOrNan() (*synapse.tests.utils.SynTest method*), 842
err() (*in module synapse.common*), 888
errinfo() (*in module synapse.common*), 888
errvar() (*synapse.lib.ast.CatchBlock method*), 619
escape() (*synapse.lib.stormtypes.LibRegx method*), 797
eth_check() (*in module synapse.lib.crypto.coin*), 572
ether_eip55() (*in module synapse.lib.crypto.coin*), 572
EthereumLib (*class in synapse.lib.stormlib.ethereum*), 591
eval() (*synapse.lib.jupyter.CmdrCore method*), 713
eval() (*synapse.lib.parser.Parser method*), 754
eval() (*synapse.lib.snap.Snap method*), 768
eval() (*synapse.lib.view.View method*), 823
evalvalu() (*synapse.lib.parser.AstConverter method*), 752
Event (*class in synapse.lib.coro*), 687
event_wait() (*in module synapse.lib.coro*), 687

exchange() (*synapse.lib.crypto.ecc.PriKey method*), 573
excinfo() (*in module synapse.common*), 888
execmain() (*synapse.lib.cell.Cell class method*), 647
execStormCmd() (*synapse.lib.storm.BackgroundCmd method*), 771
execStormCmd() (*synapse.lib.storm.BatchCmd method*), 771
execStormCmd() (*synapse.lib.storm.Cmd method*), 772
execStormCmd() (*synapse.lib.storm.CopyToCmd method*), 773
execStormCmd() (*synapse.lib.storm.CountCmd method*), 773
execStormCmd() (*synapse.lib.storm.DelNodeCmd method*), 773
execStormCmd() (*synapse.lib.storm.DiffCmd method*), 774
execStormCmd() (*synapse.lib.storm.DivertCmd method*), 774
execStormCmd() (*synapse.lib.storm.EdgesDelCmd method*), 775
execStormCmd() (*synapse.lib.storm.GraphCmd method*), 775
execStormCmd() (*synapse.lib.storm.HelpCmd method*), 776
execStormCmd() (*synapse.lib.storm.IdenCmd method*), 776
execStormCmd() (*synapse.lib.storm.IntersectCmd method*), 777
execStormCmd() (*synapse.lib.storm.LiftByVerb method*), 777
execStormCmd() (*synapse.lib.storm.LimitCmd method*), 778
execStormCmd() (*synapse.lib.storm.MaxCmd method*), 778
execStormCmd() (*synapse.lib.storm.MergeCmd method*), 779
execStormCmd() (*synapse.lib.storm.MinCmd method*), 779
execStormCmd() (*synapse.lib.storm.MoveNodesCmd method*), 780
execStormCmd() (*synapse.lib.storm.MoveTagCmd method*), 780
execStormCmd() (*synapse.lib.storm.OnceCmd method*), 781
execStormCmd() (*synapse.lib.storm.ParallelCmd method*), 781
execStormCmd() (*synapse.lib.storm.PureCmd method*), 781
execStormCmd() (*synapse.lib.storm.ReIndexCmd method*), 782
execStormCmd() (*synapse.lib.storm.RunAsCmd method*), 782
execStormCmd() (*synapse.lib.storm.ScrapeCmd method*), 785
execStormCmd() (*synapse.lib.storm.SleepCmd method*), 785
execStormCmd() (*synapse.lib.storm.SpinCmd method*), 785
execStormCmd() (*synapse.lib.storm.TagPruneCmd method*), 786
execStormCmd() (*synapse.lib.storm.TeeCmd method*), 786
execStormCmd() (*synapse.lib.storm.TreeCmd method*), 787
execStormCmd() (*synapse.lib.storm.UniqCmd method*), 787
execStormCmd() (*synapse.lib.storm.ViewExecCmd method*), 788
execStormCmd() (*synapse.lib.stormlib.cortex.StormPoolDelCmd method*), 590
execStormCmd() (*synapse.lib.stormlib.cortex.StormPoolGetCmd method*), 590
execStormCmd() (*synapse.lib.stormlib.cortex.StormPoolSetCmd method*), 591
execStormCmd() (*synapse.lib.stormlib.macro.MacroExecCmd method*), 597
execStormCmd() (*synapse.lib.stormlib.stats.StatsCountByCmd method*), 607
execStormCmd() (*synapse.tests.utils.TestCmd method*), 854
execStormTask() (*synapse.lib.storm.BackgroundCmd method*), 771
execToolMain() (*synapse.tests.utils.SynTest method*), 842
execute() (*synapse.lib.boss.Boss method*), 641
execute() (*synapse.lib.storm.Runtime method*), 783
execute() (*synapse.lib.trigger.Trigger method*), 812
executor() (*in module synapse.lib.coro*), 687
executor() (*in module synapse.lib.task*), 809
exists() (*synapse.lib.hive.Hive method*), 694
exists() (*synapse.lib.lmdbslab.MultiQueue method*), 731
exit() (*synapse.lib.cmd.Parser method*), 682
expect() (*synapse.tests.utils.TstOutPut method*), 856
ExportCmd (*class in synapse.tools.storm*), 864
exportStorm() (*synapse.cortex.CoreApi method*), 897
exportStorm() (*synapse.cortex.Cortex method*), 908
exportStormToAxon() (*synapse.cortex.Cortex method*), 908
expr_add() (*in module synapse.lib.ast*), 633
expr_div() (*in module synapse.lib.ast*), 633
expr_eq() (*in module synapse.lib.ast*), 633
expr_ge() (*in module synapse.lib.ast*), 633
expr_gt() (*in module synapse.lib.ast*), 633
expr_le() (*in module synapse.lib.ast*), 633
expr_lt() (*in module synapse.lib.ast*), 633
expr_mod() (*in module synapse.lib.ast*), 633

expr_mul() (in module synapse.lib.ast), 633
expr_ne() (in module synapse.lib.ast), 633
expr_neg() (in module synapse.lib.ast), 633
expr_not() (in module synapse.lib.ast), 633
expr_pow() (in module synapse.lib.ast), 633
expr_prefix() (in module synapse.lib.ast), 633
expr_re() (in module synapse.lib.ast), 634
expr_sub() (in module synapse.lib.ast), 634
ExprAndNode (class in synapse.lib.ast), 621
ExprDict (class in synapse.lib.ast), 622
exprdict() (synapse.lib.parser.AstConverter method), 752
ExprList (class in synapse.lib.ast), 622
exprlist() (synapse.lib.parser.AstConverter method), 752
ExprNode (class in synapse.lib.ast), 622
ExprOrNode (class in synapse.lib.ast), 622
ExtApiHandler (class in synapse.lib.httpapi), 703
extend() (synapse.lib.types.Type method), 818
extendOutpFromPatch() (synapse.tests.utils.SynTest method), 842

F

false() (synapse.tests.utils.SynTest method), 842
FatalErr, 933
FeatureNotSupported, 933
feed() (synapse.lib.link.Link method), 728
feed() (synapse.lib.msgpack.Unpk method), 740
feedBeholder() (synapse.lib.cell.Cell method), 648
feedBeholder() (synapse.lib.hiveauth.Auth method), 699
feedFromAxon() (synapse.cortex.CoreApi method), 897
feedFromAxon() (synapse.cortex.Cortex method), 908
FeedV1 (class in synapse.lib.httpapi), 704
fetch() (synapse.lib.stormlib.imap.ImapServer method), 593
FieldHelper (class in synapse.lib.types), 814
file_reporter() (synapse.utils.stormcov.plugin.StormPlugin method), 869
file_tracer() (synapse.utils.stormcov.plugin.StormPlugin method), 869
FileBase (class in synapse.models.files), 829
FileBytes (class in synapse.models.files), 829
FileExists, 933
FileModule (class in synapse.models.files), 829
FilePath (class in synapse.models.files), 830
FiltByArray (class in synapse.lib.ast), 622
filter() (synapse.lib.node.Node method), 745
FiltOper (class in synapse.lib.ast), 622
find() (synapse.lib.stormlib.xml.XmlElement method), 610
find_executable_files() (synapse.utils.stormcov.plugin.StormPlugin method), 870
find_storm_files() (synapse.utils.stormcov.plugin.StormPlugin method), 870
find_subqueries() (synapse.utils.stormcov.plugin.StormPlugin method), 870
findall() (synapse.lib.stormtypes.LibRegx method), 797
fini() (synapse.lib.base.Base method), 636
fini() (synapse.lib.base.Waiter method), 640
fini() (synapse.lib.cell.Cell method), 648
fini() (synapse.lib.lmdbslab.Slab method), 733
fini() (synapse.tests.utils.TstEnv method), 856
FiniBlock (class in synapse.lib.ast), 622
finiframe() (synapse.lib.node.Path method), 747
finiMergeTask() (synapse.lib.view.View method), 823
finiStormPool() (synapse.cortex.Cortex method), 908
finiTrigTask() (synapse.lib.view.View method), 824
fire() (synapse.lib.base.Base method), 636
firethread() (in module synapse.common), 888
first() (synapse.lib.lmdbslab.Scan method), 731
first() (synapse.lib.lmdbslab.ScanBack method), 732
first() (synapse.lib.slabseqn.SlabSeqn method), 764
firstkey() (synapse.lib.lmdbslab.Slab method), 733
FixedCache (class in synapse.lib.cache), 641
FixedCache (class in synapse.lib.stormlib.cache), 585
flatten() (in module synapse.common), 888
Float (class in synapse.lib.types), 814
FloatPacker (synapse.lib.layer.StorTypeFloat attribute), 726
FloatPackNegMax (synapse.lib.layer.StorTypeFloat attribute), 726
FloatPackNegMin (synapse.lib.layer.StorTypeFloat attribute), 726
FloatPackPosMax (synapse.lib.layer.StorTypeFloat attribute), 726
FloatPackPosMin (synapse.lib.layer.StorTypeFloat attribute), 726
fmtVersion() (in module synapse.lib.version), 821
fold() (in module synapse.lib.interval), 710
forcecommit() (synapse.lib.lmdbslab.Slab method), 733
fork() (synapse.lib.node.Path method), 747
fork() (synapse.lib.view.View method), 824
forked() (in module synapse.lib.coro), 688
ForLoop (class in synapse.lib.ast), 623
Form (class in synapse.datamodel), 926
form() (synapse.datamodel.Model method), 928
format() (synapse.lib.ast.AstNode method), 618
format() (synapse.lib.stormtypes.LibStr method), 798
format() (synapse.lib.structlog.JsonFormatter method), 809
format_component() (in module synapse.tools.healthcheck), 863
format_unescape() (in module synapse.lib.parser), 754

FormatString (*class in synapse.lib.ast*), 623
FormName (*class in synapse.lib.ast*), 623
formPhoneNode() (*in module synapse.lookup.phonenum*), 827
FormPivot (*class in synapse.lib.ast*), 623
forms (*synapse.lib.storm.Cmd attribute*), 772
forms (*synapse.tests.utils.TestCmd attribute*), 854
FormTagProp (*class in synapse.lib.ast*), 623
fpack() (*synapse.lib.layer.StorTypeFloat method*), 726
Fqdn (*class in synapse.models.inet*), 831
fqdn_check() (*in module synapse.lib.scrape*), 760
fqdn_prefix_check() (*in module synapse.lib.scrape*), 761
FREE_SPACE_CHECK_FREQ (*synapse.lib.cell.Cell attribute*), 643
fromint() (*synapse.lib.stormlib.hex.HexLib method*), 593
fromprim() (*in module synapse.lib.stormtypes*), 808
fromspawn() (*in module synapse.lib.link*), 729
fromString() (*synapse.lib.agenda.TimeUnit class method*), 612
FuncArgs (*class in synapse.lib.ast*), 623
funcargs() (*synapse.lib.parser.AstConverter method*), 752
FuncCall (*class in synapse.lib.ast*), 623
funccall() (*synapse.lib.parser.AstConverter method*), 752
Function (*class in synapse.lib.ast*), 623

G

Gate (*class in synapse.lib.stormlib.auth*), 582
ge() (*synapse.tests.utils.SynTest method*), 842
gen() (*synapse.lib.base.BaseRef method*), 639
gen() (*synapse.lib.lmdbslab.GuidStor method*), 729
genCaCert() (*synapse.lib.aha.AhaApi method*), 613
genCaCert() (*synapse.lib.aha.AhaCell method*), 616
genCaCert() (*synapse.lib.certdir.CertDir method*), 662
genCaCrl() (*synapse.lib.certdir.CertDir method*), 663
genCallsig() (*in module synapse.lib.autodoc*), 634
genClientCert() (*synapse.lib.certdir.CertDir method*), 663
genCodeCert() (*synapse.lib.certdir.CertDir method*), 663
genCrlPath() (*synapse.lib.certdir.CertDir method*), 664
gendir() (*in module synapse.common*), 888
generate() (*synapse.lib.crypto.ecc.PriKey static method*), 573
generateApiKey() (*in module synapse.lib.crypto.passwd*), 575
genFangRegex() (*in module synapse.lib.scrape*), 761
genfile() (*in module synapse.common*), 888
genGateInfo() (*synapse.lib.hiveauth.HiveRole method*), 700
genGateInfo() (*synapse.lib.hiveauth.HiveUser method*), 701
genHostCert() (*synapse.lib.certdir.CertDir method*), 664
genHostCsr() (*synapse.lib.certdir.CertDir method*), 664
genHttpSess() (*synapse.lib.cell.Cell method*), 648
genMatches() (*in module synapse.lib.scrape*), 761
genMatchesAsync() (*in module synapse.lib.scrape*), 761
genpath() (*in module synapse.common*), 889
Genr (*class in synapse.daemon*), 925
Genr (*class in synapse.telepath*), 940
genraises() (*synapse.tests.utils.SynTest method*), 843
GenrHelp (*class in synapse.lib.coro*), 687
genrhelp() (*in module synapse.lib.coro*), 688
GenrIter (*class in synapse.telepath*), 940
GenrMethod (*class in synapse.telepath*), 940
genRoleInfo() (*synapse.lib.hiveauth.AuthGate method*), 700
genTempCoreProxy() (*in module synapse.lib.jupyter*), 715
genTempStormsvcProxy() (*in module synapse.lib.jupyter*), 715
genUserCert() (*synapse.lib.certdir.CertDir method*), 665
genUserCsr() (*synapse.lib.certdir.CertDir method*), 665
genUserInfo() (*synapse.lib.hiveauth.AuthGate method*), 700
genUserOnepass() (*synapse.lib.cell.Cell method*), 648
genUserOnepass() (*synapse.lib.cell.CellApi method*), 657
GeoModule (*class in synapse.models.geospace*), 830
get() (*in module synapse.data*), 571
get() (*in module synapse.lib.scope*), 759
get() (*in module synapse.lib.thishost*), 810
get() (*synapse.axon.Axon method*), 872
get() (*synapse.axon.AxonApi method*), 879
get() (*synapse.axon.AxonHttpBySha256InvalidV1 method*), 884
get() (*synapse.axon.AxonHttpBySha256V1 method*), 885
get() (*synapse.axon.AxonHttpHasV1 method*), 885
get() (*synapse.exc.SynErr method*), 937
get() (*synapse.lib.agenda.Agenda method*), 612
get() (*synapse.lib.aha.AhaServicesV1 method*), 617
get() (*synapse.lib.base.BaseRef method*), 639
get() (*synapse.lib.boss.Boss method*), 641
get() (*synapse.lib.cache.FixedCache method*), 641
get() (*synapse.lib.cache.LruDict method*), 641
get() (*synapse.lib.cache.TagGlobs method*), 642
get() (*synapse.lib.cli.Cli method*), 679
get() (*synapse.lib.hive.Hive method*), 694

get() (*synapse.lib.hive.HiveApi method*), 695
get() (*synapse.lib.hive.HiveDict method*), 696
get() (*synapse.lib.hive.Node method*), 696
get() (*synapse.lib.hive.TeleHive method*), 697
get() (*synapse.lib.httpapi.ActiveV1 method*), 702
get() (*synapse.lib.httpapi.AuthGrantV1 method*), 702
get() (*synapse.lib.httpapi.AuthRevokeV1 method*), 702
get() (*synapse.lib.httpapi.AuthRolesV1 method*), 702
get() (*synapse.lib.httpapi.AuthRoleV1 method*), 702
get() (*synapse.lib.httpapi.AuthUsersV1 method*), 703
get() (*synapse.lib.httpapi.AuthUserV1 method*), 703
get() (*synapse.lib.httpapi.CoreInfoV1 method*), 703
get() (*synapse.lib.httpapi.ExtApiHandler method*), 703
get() (*synapse.lib.httpapi.HealthCheckV1 method*), 707
get() (*synapse.lib.httpapi.LogoutV1 method*), 707
get() (*synapse.lib.httpapi.ModelNormV1 method*), 708
get() (*synapse.lib.httpapi.ModelV1 method*), 708
get() (*synapse.lib.httpapi.ReqValidStormV1 method*), 708
get() (*synapse.lib.httpapi.RobotHandler method*), 708
get() (*synapse.lib.httpapi.StormCallV1 method*), 708
get() (*synapse.lib.httpapi.StormExportV1 method*), 709
get() (*synapse.lib.httpapi.StormNodesV1 method*), 709
get() (*synapse.lib.httpapi.StormV1 method*), 709
get() (*synapse.lib.httpapi.StormVarsGetV1 method*), 709
get() (*synapse.lib.link.Link method*), 728
get() (*synapse.lib.lmdbslab.HotCount method*), 730
get() (*synapse.lib.lmdbslab.HotKeyVal method*), 730
get() (*synapse.lib.lmdbslab.MultiQueue method*), 731
get() (*synapse.lib.lmdbslab.Slab method*), 733
get() (*synapse.lib.lmdbslab.SlabDict method*), 735
get() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
get() (*synapse.lib.node.Node method*), 745
get() (*synapse.lib.scope.Scope method*), 758
get() (*synapse.lib.slaboffs.SlabOffs method*), 764
get() (*synapse.lib.slabseqn.SlabSeqn method*), 764
get() (*synapse.lib.snap.ProtoNode method*), 766
get() (*synapse.lib.spooled.Dict method*), 770
get() (*synapse.lib.stormlib.auth.UserJson method*), 584
get() (*synapse.lib.stormlib.notifications.NotifyLib method*), 600
get() (*synapse.lib.stormlib.stats.StatTally method*), 606
get() (*synapse.lib.stormlib.xml.XmlElement method*), 610
get() (*synapse.lib.stormtypes.LibJsonStor method*), 795
get() (*synapse.lib.stormtypes.NodeProps method*), 800
get() (*synapse.lib.trigger.Trigger method*), 812
get() (*synapse.lib.trigger.Triggers method*), 812
get() (*synapse.tests.utils.HttpReflector method*), 839
get_completions() (*synapse.tools.storm.StormCompleter method*), 867
get_completions_async() (*synapse.tools.storm.StormCompleter method*), 867
get_tokens_unprocessed() (*synapse.lib.storm_format.StormLexer method*), 788
getAbrvProp() (*synapse.lib.layer.Layer method*), 720
getAddNodeOps() (*synapse.lib.snap.SnapEditor method*), 770
getAddrInfo() (*synapse.lib.link.Link method*), 728
getAddrScope() (*in module synapse.models.inet*), 833
getAddrType() (*in module synapse.models.inet*), 833
getAhaInfo() (*synapse.lib.cell.Cell method*), 648
getAhaPool() (*synapse.lib.aha.AhaApi method*), 613
getAhaPool() (*synapse.lib.aha.AhaCell method*), 616
getAhaPools() (*synapse.lib.aha.AhaApi method*), 613
getAhaPools() (*synapse.lib.aha.AhaCell method*), 616
getAhaProxy() (*in module synapse.telepath*), 942
getAhaSvc() (*synapse.lib.aha.AhaApi method*), 613
getAhaSvc() (*synapse.lib.aha.AhaCell method*), 616
getAhaSvcMirrors() (*synapse.lib.aha.AhaApi method*), 613
getAhaSvcMirrors() (*synapse.lib.aha.AhaCell method*), 616
getAhaSvcProv() (*synapse.lib.aha.AhaCell method*), 616
getAhaSvcs() (*synapse.lib.aha.AhaApi method*), 613
getAhaSvcs() (*synapse.lib.aha.AhaCell method*), 616
getAhaUrls() (*synapse.lib.aha.AhaApi method*), 613
getAhaUserEnroll() (*synapse.lib.aha.AhaCell method*), 616
getAllowedReason() (*synapse.lib.hiveauth.HiveUser method*), 701
getApiKeys() (*synapse.lib.cell.Cell method*), 648
getArgLines() (*in module synapse.lib.autodoc*), 634
getArgParseArgs() (*synapse.lib.config.Config method*), 684
getArgParser() (*in module synapse.tools.aha.easycert*), 857
getArgParser() (*in module synapse.tools.docker.validate*), 858
getArgParser() (*in module synapse.tools.json2mpk*), 863
getArgParser() (*in module synapse.tools.reload*), 864
getArgParser() (*in module synapse.tools.storm*), 867
getArgParser() (*synapse.lib.cell.Cell class method*), 648
getArgParser() (*synapse.lib.storm.BackgroundCmd method*), 771
getArgParser() (*synapse.lib.storm.BatchCmd method*), 771
getArgParser() (*synapse.lib.storm.Cmd method*), 772
getArgParser() (*synapse.lib.storm.CopyToCmd method*), 773
getArgParser() (*synapse.lib.storm.CountCmd

method), 773
getArgParser() (*synapse.lib.storm.DelNodeCmd method*), 773
getArgParser() (*synapse.lib.storm.DiffCmd method*), 774
getArgParser() (*synapse.lib.storm.DivertCmd method*), 774
getArgParser() (*synapse.lib.storm.EdgesDelCmd method*), 775
getArgParser() (*synapse.lib.storm.GraphCmd method*), 775
getArgParser() (*synapse.lib.storm.HelpCmd method*), 776
getArgParser() (*synapse.lib.storm.IdenCmd method*), 776
getArgParser() (*synapse.lib.storm.IntersectCmd method*), 777
getArgParser() (*synapse.lib.storm.LiftByVerb method*), 777
getArgParser() (*synapse.lib.storm.LimitCmd method*), 778
getArgParser() (*synapse.lib.storm.MaxCmd method*), 778
getArgParser() (*synapse.lib.storm.MergeCmd method*), 779
getArgParser() (*synapse.lib.storm.MinCmd method*), 779
getArgParser() (*synapse.lib.storm.MoveNodesCmd method*), 780
getArgParser() (*synapse.lib.storm.MoveTagCmd method*), 780
getArgParser() (*synapse.lib.storm.OnceCmd method*), 781
getArgParser() (*synapse.lib.storm.ParallelCmd method*), 781
getArgParser() (*synapse.lib.storm.PureCmd method*), 781
getArgParser() (*synapse.lib.storm.ReIndexCmd method*), 782
getArgParser() (*synapse.lib.storm.RunAsCmd method*), 782
getArgParser() (*synapse.lib.storm.ScrapeCmd method*), 785
getArgParser() (*synapse.lib.storm.SleepCmd method*), 785
getArgParser() (*synapse.lib.storm.TagPruneCmd method*), 786
getArgParser() (*synapse.lib.storm.TeeCmd method*), 786
getArgParser() (*synapse.lib.storm.TreeCmd method*), 787
getArgParser() (*synapse.lib.storm.UniqCmd method*), 787
getArgParser() (*synapse.lib.storm.ViewExecCmd method*), 788
getArgParser() (*synapse.lib.stormlib.cortex.StormPoolSetCmd method*), 591
getArgParser() (*synapse.lib.stormlib.macro.MacroExecCmd method*), 597
getArgParser() (*synapse.lib.stormlib.stats.StatsCountByCmd method*), 607
getArgParser() (*synapse.tests.utils.TestCmd method*), 854
getArgParser() (*synapse.tools.storm.ExportCmd method*), 864
getArgParser() (*synapse.tools.storm.PullFileCmd method*), 865
getArgParser() (*synapse.tools.storm.PushFileCmd method*), 865
getArgParser() (*synapse.tools.storm.RunFileCmd method*), 866
getArgParser() (*synapse.tools.storm.StormCliCmd method*), 866
getArrayPropsByType() (*synapse.datamodel.Model method*), 928
getAstText() (*synapse.lib.ast.AstNode method*), 618
getAsyncLoggerStream() (*synapse.tests.utils.SynTest method*), 843
getAuthCell() (*synapse.lib.httpapi.HandlerBase method*), 705
getAuthGate() (*synapse.lib.cell.Cell method*), 648
getAuthGate() (*synapse.lib.cell.CellApi method*), 657
getAuthGate() (*synapse.lib.hiveauth.Auth method*), 699
getAuthGates() (*synapse.lib.cell.Cell method*), 648
getAuthGates() (*synapse.lib.cell.CellApi method*), 657
getAuthGates() (*synapse.lib.hiveauth.Auth method*), 699
getAuthInfo() (*synapse.lib.cell.CellApi method*), 657
getAuthRoles() (*synapse.lib.cell.Cell method*), 648
getAuthRoles() (*synapse.lib.cell.CellApi method*), 657
getAuthUsers() (*synapse.lib.cell.Cell method*), 648
getAuthUsers() (*synapse.lib.cell.CellApi method*), 657
getAvailableMemory() (*in module synapse.lib.platforms.linux*), 580
getAxon() (*synapse.axon.AxonHandlerMixin method*), 884
getAxon() (*synapse.cortex.Cortex method*), 908
getAxon() (*synapse.cortex.CortexAxonMixin method*), 922
getAxonBytes() (*synapse.cortex.CoreApi method*), 897
getAxonInfo() (*synapse.axon.AxonFileHandler method*), 884
getAxonInfo() (*synapse.cortex.CortexAxonMixin method*), 922
getAxonUpload() (*synapse.cortex.CoreApi method*), 897
getBackupInfo() (*synapse.lib.cell.Cell method*), 649

getBackupInfo() (*synapse.lib.cell.CellApi method*), 657
getBackups() (*synapse.lib.cell.Cell method*), 649
getBackups() (*synapse.lib.cell.CellApi method*), 658
getByIndxByts() (*synapse.lib.slabseqn.SlabSeqn method*), 764
getByLayer() (*synapse.lib.node.Node method*), 746
getbytes() (*in module synapse.common*), 890
getCaCert() (*synapse.lib.aha.AhaApi method*), 613
getCaCert() (*synapse.lib.aha.AhaCell method*), 616
getCaCert() (*synapse.lib.aha.EnrollApi method*), 617
getCaCert() (*synapse.lib.aha.ProvApi method*), 617
getCaCert() (*synapse.lib.certdir.CertDir method*), 665
getCaCertBytes() (*synapse.lib.certdir.CertDir method*), 666
getCaCertPath() (*synapse.lib.certdir.CertDir method*), 666
getCaCerts() (*synapse.lib.certdir.CertDir method*), 666
getCachedSslCtx() (*synapse.lib.cell.Cell method*), 649
getCaKey() (*synapse.lib.certdir.CertDir method*), 666
getCaKeyPath() (*synapse.lib.certdir.CertDir method*), 667
getCallSig() (*in module synapse.lib.stormtypes*), 808
getCatchBlock() (*synapse.lib.ast.TryCatch method*), 632
getCell() (*in module synapse.lib.rstorm*), 757
getCellApi() (*synapse.cortex.Cortex method*), 908
getCellApi() (*synapse.cryotank.CryoCell method*), 923
getCellApi() (*synapse.lib.cell.Cell method*), 649
getCellIden() (*synapse.lib.cell.Cell method*), 649
getCellIden() (*synapse.lib.cell.CellApi method*), 658
getCellIden() (*synapse.lib.view.ViewApi method*), 826
getCellInfo() (*synapse.axon.Axon method*), 872
getCellInfo() (*synapse.lib.cell.Cell method*), 649
getCellInfo() (*synapse.lib.cell.CellApi method*), 658
getCellInfo() (*synapse.tests.utils.ReloadCell method*), 840
getCellNexsRoot() (*synapse.lib.cell.Cell method*), 649
getCellRunId() (*synapse.lib.cell.Cell method*), 649
getCellRunId() (*synapse.lib.cell.CellApi method*), 658
getCellType() (*synapse.lib.cell.Cell class method*), 649
getCellType() (*synapse.lib.cell.CellApi method*), 658
getCellUser() (*synapse.lib.cell.CellApi method*), 658
getCertDir() (*in module synapse.lib.certdir*), 677
getCertDirn() (*in module synapse.lib.certdir*), 677
getChangeDist() (*synapse.lib.nexus.NexsRoot method*), 743
getCidrRange() (*synapse.models.inet.IPv4 method*), 831
getCidrRange() (*synapse.models.inet.IPv6 method*), 832
getClientCert() (*synapse.lib.certdir.CertDir method*), 667
getClientCertPath() (*synapse.lib.certdir.CertDir method*), 667
getClientSSLContext() (*synapse.lib.certdir.CertDir method*), 668
getClsNames() (*in module synapse.lib.reflect*), 755
getCmdBrief() (*synapse.lib.cli.Cmd method*), 680
getCmdBrief() (*synapse.lib.storm.Cmd class method*), 772
getCmdByName() (*synapse.lib.cli.Cli method*), 679
getCmdDoc() (*synapse.lib.cli.Cmd method*), 680
getCmdItem() (*synapse.lib.cli.Cmd method*), 680
getCmdlineMapping() (*synapse.lib.config.Config method*), 684
getCmdName() (*synapse.lib.cli.Cmd method*), 680
getCmdNames() (*synapse.lib.cli.Cli method*), 679
getCmdOpts() (*synapse.lib.cli.Cmd method*), 680
getCmdOpts() (*synapse.tools.storm.StormCliCmd method*), 866
getCmdPrompt() (*synapse.lib.cli.Cli method*), 679
getCmdRuntime() (*synapse.lib.storm.Runtime method*), 783
getCmprCtor() (*synapse.lib.types.Type method*), 818
getCodeCert() (*synapse.lib.certdir.CertDir method*), 668
getCodeCertPath() (*synapse.lib.certdir.CertDir method*), 668
getCodeKey() (*synapse.lib.certdir.CertDir method*), 668
getCodeKeyPath() (*synapse.lib.certdir.CertDir method*), 668
getCompOffs() (*synapse.datamodel.Prop method*), 929
getCompOffs() (*synapse.lib.types.Comp method*), 813
getCompOffs() (*synapse.lib.types.Edge method*), 814
getCompOffs() (*synapse.lib.types.TimeEdge method*), 818
getCompOffs() (*synapse.lib.types.Type method*), 818
getCondEval() (*synapse.lib.ast.AbsPropCond method*), 617
getCondEval() (*synapse.lib.ast.AndCond method*), 617
getCondEval() (*synapse.lib.ast.ArrayCond method*), 618
getCondEval() (*synapse.lib.ast.HasAbsPropCond method*), 624
getCondEval() (*synapse.lib.ast.HasRelPropCond method*), 624
getCondEval() (*synapse.lib.ast.HasTagPropCond method*), 624
getCondEval() (*synapse.lib.ast.NotCond method*), 627
getCondEval() (*synapse.lib.ast.OrCond method*), 627
getCondEval() (*synapse.lib.ast.RelPropCond method*), 628
getCondEval() (*synapse.lib.ast.SubqCond method*),

630
getCondEval() (*synapse.lib.ast.TagCond method*), 631
getCondEval() (*synapse.lib.ast.TagPropCond method*), 631
getCondEval() (*synapse.lib.ast.TagValuCond method*), 631
getCondEval() (*synapse.lib.ast.Value method*), 632
getConfFromCell() (*synapse.lib.config.Config class method*), 684
getConfOpt() (*synapse.lib.cell.Cell method*), 649
getConfPath() (*synapse.lib.module.CoreModule method*), 738
getCore() (*synapse.lib.httpapi.StormHandler method*), 709
getCoreInfo() (*synapse.cortex.CoreApi method*), 897
getCoreInfo() (*synapse.cortex.Cortex method*), 908
getCoreInfoV2() (*synapse.cortex.CoreApi method*), 897
getCoreInfoV2() (*synapse.cortex.Cortex method*), 908
getCoreMod() (*synapse.cortex.Cortex method*), 908
getCoreMods() (*synapse.cortex.CoreApi method*), 897
getCoreMods() (*synapse.cortex.Cortex method*), 908
getCoreQueue() (*synapse.cortex.Cortex method*), 909
getCosignSignature() (*in module synapse.tools.docker.validate*), 858
getCrlPath() (*synapse.lib.certdir.CertDir method*), 668
getCurrentLockedMemory() (*in module synapse.lib.platforms.linux*), 580
getCustomHeaders() (*synapse.lib.httpapi.HandlerBase method*), 705
getData() (*synapse.lib.node.Node method*), 746
getData() (*synapse.lib.snap.ProtoNode method*), 766
getDataModel() (*synapse.cortex.Cortex method*), 909
getDeprLocks() (*synapse.cortex.Cortex method*), 909
getDescr() (*synapse.lib.storm.Cmd method*), 772
getDescr() (*synapse.lib.storm.PureCmd method*), 781
getDiagInfo() (*synapse.lib.cell.CellApi method*), 658
getDirSize() (*in module synapse.common*), 889
getDmon() (*synapse.lib.storm.DmonManager method*), 774
getDmonDef() (*synapse.lib.storm.DmonManager method*), 774
getDmonDefs() (*synapse.lib.storm.DmonManager method*), 774
getDmonRunlog() (*synapse.lib.storm.DmonManager method*), 774
getDmonSessions() (*synapse.lib.cell.Cell method*), 649
getDmonSessions() (*synapse.lib.cell.CellApi method*), 658
getDmonUser() (*synapse.lib.cell.Cell method*), 650
getDoc() (*in module synapse.lib.stormtypes*), 808
getDocData() (*in module synapse.lib.jupyter*), 715
getDocPath() (*in module synapse.lib.jupyter*), 715
getDynLocal() (*in module synapse.lib.dyndeps*), 689
getDynMeth() (*in module synapse.lib.dyndeps*), 689
getDynMod() (*in module synapse.lib.dyndeps*), 689
getEdgeGlobRegx() (*in module synapse.lib.cache*), 642
getEdges() (*synapse.lib.layer.Layer method*), 720
getEdges() (*synapse.lib.stormtypes.Layer method*), 791
getEdges() (*synapse.lib.view.View method*), 824
getEdgesByN1() (*synapse.lib.stormtypes.Layer method*), 791
getEdgesByN2() (*synapse.lib.stormtypes.Layer method*), 791
getEdgeVerbs() (*synapse.lib.layer.Layer method*), 720
getEdgeVerbs() (*synapse.lib.view.View method*), 824
getEditIndx() (*synapse.lib.layer.Layer method*), 720
getEditIndx() (*synapse.lib.layer.LayerApi method*), 725
getEditOffs() (*synapse.lib.layer.Layer method*), 720
getEditor() (*synapse.lib.snap.Snap method*), 768
getEditSize() (*synapse.lib.layer.Layer method*), 720
getEditSize() (*synapse.lib.layer.LayerApi method*), 725
getEditSize() (*synapse.lib.view.ViewApi method*), 826
getEmbeds() (*synapse.lib.node.Node method*), 746
getEnvarMapping() (*synapse.lib.config.Config method*), 684
getEnvPrefix() (*synapse.cryotank.CryoCell class method*), 923
getEnvPrefix() (*synapse.lib.aha.AhaCell class method*), 616
getEnvPrefix() (*synapse.lib.cell.Cell class method*), 650
getEnvPrefix() (*synapse.lib.jsonstor.JsonStorCell class method*), 713
getErrValu() (*synapse.lib.ast.TryCatch method*), 632
getExtModel() (*synapse.cortex.Cortex method*), 909
getExtModel() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
getFeedFunc() (*synapse.cortex.Cortex method*), 909
getFeedFuncs() (*synapse.cortex.CoreApi method*), 897
getFeedFuncs() (*synapse.cortex.Cortex method*), 909
getfile() (*in module synapse.common*), 890
getFileMappedRegion() (*in module synapse.lib.platforms.linux*), 580
getFlatEdits() (*in module synapse.lib.layer*), 728
getFormCounts() (*synapse.cortex.Cortex method*), 909
getFormCounts() (*synapse.lib.layer.Layer method*), 720
getFormCounts() (*synapse.lib.view.View method*), 824
getFormDef() (*synapse.datamodel.Form method*), 926
getFormProps() (*synapse.lib.layer.Layer method*), 720
getForms() (*in module synapse.lib.scrape*), 762
getFormsByPrefix() (*synapse.datamodel.Model method*), 928
getGcInfo() (*synapse.lib.cell.CellApi method*), 658

getGraph() (*synapse.lib.storm.Runtime method*), 783
getHealthCheck() (*synapse.lib.cell.Cell method*), 650
getHealthCheck() (*synapse.lib.cell.CellApi method*), 658
getHierIndx() (*synapse.lib.layer.StorTypeHier method*), 727
getHiveAuth() (*synapse.lib.hive.Hive method*), 694
getHiveKey() (*synapse.lib.cell.Cell method*), 650
getHiveKey() (*synapse.lib.cell.CellApi method*), 658
getHiveKeys() (*synapse.lib.cell.Cell method*), 650
getHiveKeys() (*synapse.lib.cell.CellApi method*), 658
getHostCaPath() (*synapse.lib.certdir.CertDir method*), 668
getHostCert() (*synapse.lib.certdir.CertDir method*), 668
getHostCertHash() (*synapse.lib.certdir.CertDir method*), 669
getHostCertPath() (*synapse.lib.certdir.CertDir method*), 669
getHostCsrPath() (*synapse.lib.certdir.CertDir method*), 669
getHostKey() (*synapse.lib.certdir.CertDir method*), 669
getHostKeyPath() (*synapse.lib.certdir.CertDir method*), 669
getHotCount() (*synapse.lib.lmdbslab.Slab method*), 733
getHttpApi() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
getHttpApiByPath() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
getHttpExtApi() (*synapse.cortex.Cortex method*), 909
getHttpExtApiByPath() (*synapse.cortex.CoreApi method*), 898
getHttpExtApiByPath() (*synapse.cortex.Cortex method*), 909
getHttpExtApis() (*synapse.cortex.Cortex method*), 909
getHttpSess() (*synapse.tests.utils.SynTest method*), 843
getHttpSessDict() (*synapse.lib.cell.Cell method*), 650
getHugeIndx() (*synapse.lib.layer.StorTypeHugeNum method*), 727
getIden() (*synapse.lib.layer.LayerApi method*), 725
getIdenFutu() (*synapse.lib.layer.Layer method*), 720
getInput() (*synapse.lib.storm.Runtime method*), 783
getIntIndx() (*synapse.lib.layer.StorTypeInt method*), 727
getIPv6Indx() (*synapse.lib.layer.StorTypeIpv6 method*), 727
getItemCmdr() (*in module synapse.lib.cmdr*), 682
getItemCmdr() (*in module synapse.lib.jupyter*), 716
getItemLocals() (*in module synapse.lib.reflect*), 756
getItems() (*in module synapse.tools.feed*), 861
getItemStorm() (*in module synapse.lib.jupyter*), 716
getJSON() (*in module synapse.data*), 572
getJsonBody() (*synapse.lib.httpapi.HandlerBase method*), 705
getJsonObj() (*synapse.cortex.Cortex method*), 909
getJsonObjProp() (*synapse.cortex.Cortex method*), 909
getJsonObjs() (*synapse.cortex.Cortex method*), 909
getJsSchema() (*in module synapse.lib.config*), 686
getJsValidator() (*in module synapse.lib.config*), 686
getLangCodes() (*in module synapse.lookup.pe*), 827
getLayer() (*synapse.cortex.Cortex method*), 909
getLayerDef() (*synapse.cortex.Cortex method*), 909
getLayerDefs() (*synapse.cortex.Cortex method*), 909
getLayerSize() (*synapse.lib.layer.Layer method*), 720
getLibC() (*in module synapse.lib.platforms.common*), 579
getLibC() (*in module synapse.lib.platforms.windows*), 581
getLibDocs() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
getLiftHintCmpr() (*synapse.lib.types.Type method*), 819
getLiftHintCmprCtor() (*synapse.lib.types.Type method*), 819
getLiftHints() (*synapse.lib.ast.AndCond method*), 617
getLiftHints() (*synapse.lib.ast.FiltOper method*), 622
getLiftHints() (*synapse.lib.ast.HasRelPropCond method*), 624
getLiftHints() (*synapse.lib.ast.RelPropCond method*), 628
getLiftHints() (*synapse.lib.ast.TagCond method*), 631
getLiftHints() (*synapse.lib.ast.Value method*), 632
getLink() (*in module synapse.lib.autodoc*), 634
getLoadCmdTypes() (*in module synapse.lookup.macho*), 827
getLocalProxy() (*synapse.lib.cell.Cell method*), 650
getLocalUrl() (*synapse.lib.cell.Cell method*), 650
getLogExtra() (*synapse.lib.cell.Cell method*), 650
getLoggerStream() (*synapse.tests.utils.SynTest method*), 844
getMagicPromptColors() (*synapse.tests.utils.SynTest method*), 845
getMagicPromptLines() (*synapse.tests.utils.SynTest method*), 845
getMaxHotFixes() (*in module synapse.lib.stormlib.cell*), 586
getMaxLockedMemory() (*in module synapse.lib.platforms.linux*), 580
getMergeRequest() (*synapse.lib.stormtypes.View method*), 807
getMergeRequest() (*synapse.lib.view.View method*),

824
getMergeRequestSummary() (*synapse.lib.stormtypes.View method*), 807
getMerges() (*synapse.lib.stormtypes.View method*), 807
getMerges() (*synapse.lib.view.View method*), 824
getMergeVotes() (*synapse.lib.view.View method*), 824
getMergingViews() (*synapse.lib.stormtypes.View method*), 807
getMergingViews() (*synapse.lib.view.View method*), 824
getMethName() (*in module synapse.lib.reflect*), 756
getMirrorStatus() (*synapse.lib.layer.Layer method*), 720
getMirrorStatus() (*synapse.lib.stormtypes.Layer method*), 791
getMirrorUrls() (*synapse.lib.cell.Cell method*), 650
getMirrorUrls() (*synapse.lib.cell.CellApi method*), 658
getModDir() (*synapse.lib.module.CoreModule method*), 738
getModelDefs() (*synapse.cortex.CoreApi method*), 898
getModelDefs() (*synapse.cortex.Cortex method*), 909
getModelDefs() (*synapse.datamodel.Model method*), 928
getModelDefs() (*synapse.lib.module.CoreModule method*), 739
getModelDefs() (*synapse.models.auth.AuthModule method*), 828
getModelDefs() (*synapse.models.base.BaseModule method*), 828
getModelDefs() (*synapse.models.belief.BeliefModule method*), 828
getModelDefs() (*synapse.models.biz.BizModule method*), 829
getModelDefs() (*synapse.models.crypto.CryptoModule method*), 829
getModelDefs() (*synapse.models.dns.DnsModule method*), 829
getModelDefs() (*synapse.models.economic.EconModule method*), 829
getModelDefs() (*synapse.models.files.FileModule method*), 829
getModelDefs() (*synapse.models.geopol.PolModule method*), 830
getModelDefs() (*synapse.models.geospace.GeoModule method*), 830
getModelDefs() (*synapse.models.gov.cn.GovCnModule method*), 828
getModelDefs() (*synapse.models.gov.intl.GovIntlModule method*), 828
getModelDefs() (*synapse.models.gov.us.GovUsModule method*), 828
getModelDefs() (*synapse.models.inet.InetModule method*), 832
getModelDefs() (*synapse.models.infotech.ItModule method*), 833
getModelDefs() (*synapse.models.language.LangModule method*), 834
getModelDefs() (*synapse.models.material.MatModule method*), 835
getModelDefs() (*synapse.models.media.MediaModule method*), 835
getModelDefs() (*synapse.models.orgs.OuModule method*), 835
getModelDefs() (*synapse.models.person.PsModule method*), 835
getModelDefs() (*synapse.models.proj.ProjectModule method*), 835
getModelDefs() (*synapse.models.risk.RiskModule method*), 836
getModelDefs() (*synapse.models.science.ScienceModule method*), 836
getModelDefs() (*synapse.models.syn.SynModule method*), 836
getModelDefs() (*synapse.models.telco.TelcoModule method*), 837
getModelDefs() (*synapse.models.transport.TransportModule method*), 838
getModelDefs() (*synapse.tests.utils.DeprModule method*), 839
getModelDefs() (*synapse.tests.utils.TestModule method*), 854
getModelDict() (*synapse.cortex.CoreApi method*), 898
getModelDict() (*synapse.cortex.Cortex method*), 909
getModelDict() (*synapse.datamodel.Model method*), 928
getModelVers() (*synapse.lib.layer.Layer method*), 720
getModName() (*synapse.lib.module.CoreModule method*), 738
getModPath() (*synapse.lib.module.CoreModule method*), 738
getModRuntime() (*synapse.lib.storm.Runtime method*), 783
getMultiQueue() (*synapse.lib.lmdbslab.Slab method*), 733
getName() (*synapse.lib.storm.Cmd method*), 772
getName() (*synapse.lib.storm.PureCmd method*), 782
getNameAbrv() (*synapse.lib.lmdbslab.Slab method*), 733
getNetRange() (*synapse.models.inet.IPv4 method*), 831
getNetRange() (*synapse.models.inet.IPv6 method*), 832
getNexsIndx() (*synapse.lib.cell.Cell method*), 650
getNexsIndx() (*synapse.lib.cell.CellApi method*), 658
getNextBootUrl() (*synapse.telepath.ClientV2 method*), 940
getNexusChanges() (*synapse.lib.cell.Cell method*), 650
getNexusChanges() (*synapse.lib.cell.CellApi method*), 658

getNodeByBuid() (*synapse.lib.snap.Snap method*), 768
getNodeByBuid() (*synapse.lib.snap.SnapEditor method*), 770
getNodeByNdef() (*synapse.cortex.Cortex method*), 909
getNodeByNdef() (*synapse.lib.snap.Snap method*), 768
getNodeData() (*synapse.lib.layer.Layer method*), 720
getNodeData() (*synapse.lib.snap.Snap method*), 768
getNodeEdit() (*synapse.lib.snap.ProtoNode method*), 766
getNodeEditor() (*synapse.lib.snap.Snap method*), 768
getNodeEditPerms() (*in module synapse.lib.layer*), 728
getNodeEdits() (*synapse.lib.snap.SnapEditor method*), 770
getNodeEditWindow() (*synapse.lib.layer.Layer method*), 721
getNodeForm() (*synapse.lib.layer.Layer method*), 721
getNodeRefs() (*synapse.lib.node.Node method*), 746
getNodeTag() (*synapse.lib.layer.Layer method*), 721
getNodeValu() (*synapse.lib.layer.IndxBy method*), 719
getNodeValu() (*synapse.lib.layer.IndxByForm method*), 719
getNodeValu() (*synapse.lib.layer.IndxByProp method*), 720
getNodeValu() (*synapse.lib.layer.IndxByPropArray method*), 720
getNodeValu() (*synapse.lib.layer.IndxByTagProp method*), 720
getNodeValu() (*synapse.lib.layer.Layer method*), 721
getNodeValuForm() (*synapse.lib.layer.IndxByTag method*), 720
getOAuthAccessToken() (*synapse.lib.oauth.OAuthMixin method*), 751
getOAuthClient() (*synapse.lib.oauth.OAuthMixin method*), 751
getOAuthProvider() (*synapse.lib.oauth.OAuthMixin method*), 751
getObjLocals() (*synapse.lib.stormhttp.HttpResp method*), 789
getObjLocals() (*synapse.lib.stormhttp.LibHttp method*), 789
getObjLocals() (*synapse.lib.stormhttp.WebSocket method*), 789
getObjLocals() (*synapse.lib.stormlib.aha.AhaLib method*), 581
getObjLocals() (*synapse.lib.stormlib.aha.AhaPoolLib method*), 581
getObjLocals() (*synapse.lib.stormlib.auth.LibAuth method*), 582
getObjLocals() (*synapse.lib.stormlib.auth.LibGates method*), 582
getObjLocals() (*synapse.lib.stormlib.auth.LibRoles method*), 582
getObjLocals() (*synapse.lib.stormlib.auth.LibUser method*), 583
getObjLocals() (*synapse.lib.stormlib.auth.LibUsers method*), 583
getObjLocals() (*synapse.lib.stormlib.auth.Role method*), 583
getObjLocals() (*synapse.lib.stormlib.auth.User method*), 583
getObjLocals() (*synapse.lib.stormlib.backup.BackupLib method*), 585
getObjLocals() (*synapse.lib.stormlib.basex.BaseXLib method*), 585
getObjLocals() (*synapse.lib.stormlib.cache.FixedCache method*), 585
getObjLocals() (*synapse.lib.stormlib.cache.LibCache method*), 585
getObjLocals() (*synapse.lib.stormlib.cell.CellLib method*), 586
getObjLocals() (*synapse.lib.stormlib.compression.Bzip2Lib method*), 586
getObjLocals() (*synapse.lib.stormlib.compression.GzipLib method*), 586
getObjLocals() (*synapse.lib.stormlib.compression.ZlibLib method*), 587
getObjLocals() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
getObjLocals() (*synapse.lib.stormlib.cortex.HttpApi method*), 588
getObjLocals() (*synapse.lib.stormlib.cortex.HttpReq method*), 590
getObjLocals() (*synapse.lib.stormlib.easyperm.LibEasyPerm method*), 591
getObjLocals() (*synapse.lib.stormlib.ethereum.EthereumLib method*), 591
getObjLocals() (*synapse.lib.stormlib.gis.GisLib method*), 592
getObjLocals() (*synapse.lib.stormlib.graph.GraphLib method*), 592
getObjLocals() (*synapse.lib.stormlib.hashes.LibHashes method*), 592
getObjLocals() (*synapse.lib.stormlib.hashes.LibHmac method*), 592
getObjLocals() (*synapse.lib.stormlib.hex.HexLib method*), 593
getObjLocals() (*synapse.lib.stormlib.imap.ImapLib method*), 593
getObjLocals() (*synapse.lib.stormlib.imap.ImapServer method*), 593
getObjLocals() (*synapse.lib.stormlib.infosec.CvssLib method*), 594
getObjLocals() (*synapse.lib.stormlib.infosec.MitreAttackFlowLib method*), 595
getObjLocals() (*synapse.lib.stormlib.ipv6.LibIpv6 method*), 595

getObjLocals() (*synapse.lib.stormlib.iters.LibIters method*), 595
getObjLocals() (*synapse.lib.stormlib.json.JsonLib method*), 596
getObjLocals() (*synapse.lib.stormlib.json.JsonSchema method*), 596
getObjLocals() (*synapse.lib.stormlib.log.LoggerLib method*), 596
getObjLocals() (*synapse.lib.stormlib.macro.LibMacro method*), 597
getObjLocals() (*synapse.lib.stormlib.math.MathLib method*), 597
getObjLocals() (*synapse.lib.stormlib.mime.LibMimeHtml method*), 598
getObjLocals() (*synapse.lib.stormlib.model.LibModel method*), 598
getObjLocals() (*synapse.lib.stormlib.model.LibModelDeprecated method*), 598
getObjLocals() (*synapse.lib.stormlib.model.LibModelEdge method*), 598
getObjLocals() (*synapse.lib.stormlib.model.LibModelTags method*), 599
getObjLocals() (*synapse.lib.stormlib.model.ModelForm method*), 599
getObjLocals() (*synapse.lib.stormlib.model.ModelType method*), 599
getObjLocals() (*synapse.lib.stormlib.modelext.LibModelExt method*), 600
getObjLocals() (*synapse.lib.stormlib.notifications.NotifyLib method*), 600
getObjLocals() (*synapse.lib.stormlib.oauth.OAuthV1Client method*), 601
getObjLocals() (*synapse.lib.stormlib.oauth.OAuthV1Lib method*), 601
getObjLocals() (*synapse.lib.stormlib.oauth.OAuthV2Lib method*), 601
getObjLocals() (*synapse.lib.stormlib.pack.LibPack method*), 601
getObjLocals() (*synapse.lib.stormlib.project.LibProjects method*), 602
getObjLocals() (*synapse.lib.stormlib.project.ProjectEpics method*), 602
getObjLocals() (*synapse.lib.stormlib.project.ProjectSprints method*), 603
getObjLocals() (*synapse.lib.stormlib.project.ProjectTicketComment method*), 603
getObjLocals() (*synapse.lib.stormlib.project.ProjectTicketComments method*), 603
getObjLocals() (*synapse.lib.stormlib.project.ProjectTickets method*), 604
getObjLocals() (*synapse.lib.stormlib.random.LibRandom method*), 604
getObjLocals() (*synapse.lib.stormlib.scrape.LibScrape method*), 604
getObjLocals() (*synapse.lib.stormlib.smtp.SmtpLib method*), 605
getObjLocals() (*synapse.lib.stormlib.spooled.LibSpooled method*), 605
getObjLocals() (*synapse.lib.stormlib.stats.LibStats method*), 606
getObjLocals() (*synapse.lib.stormlib.stats.StatTally method*), 606
getObjLocals() (*synapse.lib.stormlib.stix.LibStix method*), 607
getObjLocals() (*synapse.lib.stormlib.stix.LibStixExport method*), 607
getObjLocals() (*synapse.lib.stormlib.stix.LibStixImport method*), 608
getObjLocals() (*synapse.lib.stormlib.stix.StixBundle method*), 608
getObjLocals() (*synapse.lib.stormlib.storm.LibStorm method*), 608
getObjLocals() (*synapse.lib.stormlib.vault.LibVault method*), 609
getObjLocals() (*synapse.lib.stormlib.vault.Vault method*), 609
getObjLocals() (*synapse.lib.stormlib.version.VersionLib method*), 610
getObjLocals() (*synapse.lib.stormlib.xml.LibXml method*), 610
getObjLocals() (*synapse.lib.stormlib.yaml.LibYaml method*), 610
getObjLocals() (*synapse.lib.stormtypes.Bytes method*), 790
getObjLocals() (*synapse.lib.stormtypes.CronJob method*), 791
getObjLocals() (*synapse.lib.stormtypes.Layer method*), 791
getObjLocals() (*synapse.lib.stormtypes.LibAxon method*), 792
getObjLocals() (*synapse.lib.stormtypes.LibBase method*), 793
getObjLocals() (*synapse.lib.stormtypes.LibBase64 method*), 793
getObjLocals() (*synapse.lib.stormtypes.LibBytes method*), 793
getObjLocals() (*synapse.lib.stormtypes.LibCron method*), 794
getObjLocals() (*synapse.lib.stormtypes.LibCsv method*), 794
getObjLocals() (*synapse.lib.stormtypes.LibDict method*), 794
getObjLocals() (*synapse.lib.stormtypes.LibDmon method*), 794
getObjLocals() (*synapse.lib.stormtypes.LibExport method*), 794
getObjLocals() (*synapse.lib.stormtypes.LibFeed method*), 795

getObjLocals() (*synapse.lib.stormtypes.LibGlobals method*), 795
getObjLocals() (*synapse.lib.stormtypes.LibLayer method*), 796
getObjLocals() (*synapse.lib.stormtypes.LibLift method*), 796
getObjLocals() (*synapse.lib.stormtypes.LibPipe method*), 796
getObjLocals() (*synapse.lib.stormtypes.LibPkg method*), 796
getObjLocals() (*synapse.lib.stormtypes.LibPs method*), 796
getObjLocals() (*synapse.lib.stormtypes.LibQueue method*), 797
getObjLocals() (*synapse.lib.stormtypes.LibRegx method*), 797
getObjLocals() (*synapse.lib.stormtypes.LibService method*), 797
getObjLocals() (*synapse.lib.stormtypes.LibStr method*), 798
getObjLocals() (*synapse.lib.stormtypes.LibTags method*), 798
getObjLocals() (*synapse.lib.stormtypes.LibTelepath method*), 798
getObjLocals() (*synapse.lib.stormtypes.LibTime method*), 798
getObjLocals() (*synapse.lib.stormtypes.LibTrigger method*), 799
getObjLocals() (*synapse.lib.stormtypes.LibVars method*), 799
getObjLocals() (*synapse.lib.stormtypes.LibView method*), 799
getObjLocals() (*synapse.lib.stormtypes.List method*), 800
getObjLocals() (*synapse.lib.stormtypes.Node method*), 800
getObjLocals() (*synapse.lib.stormtypes.NodeData method*), 800
getObjLocals() (*synapse.lib.stormtypes.NodeProps method*), 801
getObjLocals() (*synapse.lib.stormtypes.Number method*), 801
getObjLocals() (*synapse.lib.stormtypes.Path method*), 801
getObjLocals() (*synapse.lib.stormtypes.Pipe method*), 802
getObjLocals() (*synapse.lib.stormtypes.Query method*), 803
getObjLocals() (*synapse.lib.stormtypes.Queue method*), 804
getObjLocals() (*synapse.lib.stormtypes.Set method*), 804
getObjLocals() (*synapse.lib.stormtypes.StormHiveDict method*), 804
getObjLocals() (*synapse.lib.stormtypes.StormType method*), 805
getObjLocals() (*synapse.lib.stormtypes.Str method*), 806
getObjLocals() (*synapse.lib.stormtypes.Text method*), 806
getObjLocals() (*synapse.lib.stormtypes.Trigger method*), 806
getObjLocals() (*synapse.lib.stormtypes.View method*), 807
getObjLocals() (*synapse.lib.stormwhois.LibWhois method*), 809
getOffsetEvent() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
getOffsetEvent() (*synapse.lib.slabseqn.SlabSeqn method*), 765
getOneNode() (*synapse.lib.storm.Runtime method*), 783
getOpt() (*synapse.lib.storm.Runtime method*), 783
getPathList() (*synapse.lib.jsonstor.JsonStor method*), 711
getPathList() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getPathList() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getPathObj() (*synapse.lib.jsonstor.JsonStor method*), 711
getPathObj() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getPathObj() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getPathObjProp() (*synapse.lib.jsonstor.JsonStor method*), 711
getPathObjProp() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getPathObjProp() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getPathObjs() (*synapse.lib.jsonstor.JsonStor method*), 711
getPathObjs() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getPathObjs() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getPbkdf2() (*in module synapse.lib.crypto.passwd*), 575
getPermDef() (*synapse.lib.cell.Cell method*), 650
getPermDef() (*synapse.lib.cell.CellApi method*), 658
getPermDef() (*synapse.lib.stormlib.auth.LibAuth method*), 582
getPermDefs() (*synapse.lib.cell.Cell method*), 650
getPermDefs() (*synapse.lib.cell.CellApi method*), 658
getPermDefs() (*synapse.lib.stormlib.auth.LibAuth method*), 582
getPhoneInfo() (*in module synapse.lookup.phonenum*), 827

getPipeline() (*synapse.telepath.Proxy method*), 941
getPivsIn() (*synapse.lib.ast.PivotIn method*), 627
getPivsOut() (*synapse.lib.ast.PivotOut method*), 627
getPoolLink() (*synapse.telepath.Proxy method*), 941
getPosInfo() (*synapse.lib.ast.AstNode method*), 618
getPropAbrv() (*synapse.lib.layer.Layer method*), 721
getPropAndValu() (*synapse.lib.ast.PropValue method*), 628
getPropArrayCount() (*synapse.lib.layer.Layer method*), 721
getPropArrayCount() (*synapse.lib.view.View method*), 824
getPropArrayValuCount() (*synapse.lib.layer.Layer method*), 721
getPropCount() (*synapse.lib.layer.Layer method*), 721
getPropCount() (*synapse.lib.view.View method*), 824
getPropDef() (*synapse.datamodel.Prop method*), 929
getPropNorm() (*synapse.cortex.CoreApi method*), 898
getPropNorm() (*synapse.cortex.Cortex method*), 909
getProps() (*synapse.datamodel.Model method*), 928
getPropsByType() (*synapse.datamodel.Model method*), 928
getPropValuCount() (*synapse.lib.layer.Layer method*), 721
getProvInfo() (*synapse.lib.aha.ProvApi method*), 617
getraw() (*synapse.lib.slabseqn.SlabSeqn method*), 765
getRefsOut() (*synapse.datamodel.Form method*), 926
getRegrAxon() (*synapse.tests.utils.SynTest method*), 845
getRegrCore() (*synapse.tests.utils.SynTest method*), 845
getRegrDir() (*synapse.tests.utils.SynTest method*), 845
getReloadableSystems() (*synapse.lib.cell.Cell method*), 650
getReloadableSystems() (*synapse.lib.cell.CellApi method*), 658
getReturnLines() (*in module synapse.lib.autodoc*), 634
getRightHints() (*synapse.lib.ast.LiftProp method*), 625
getRoleByName() (*synapse.lib.hiveauth.Auth method*), 699
getRoleDef() (*synapse.lib.cell.Cell method*), 650
getRoleDef() (*synapse.lib.cell.CellApi method*), 658
getRoleDefByName() (*synapse.lib.cell.Cell method*), 650
getRoleDefByName() (*synapse.lib.cell.CellApi method*), 658
getRoleDefs() (*synapse.lib.cell.Cell method*), 650
getRoleDefs() (*synapse.lib.cell.CellApi method*), 658
getRoleInfo() (*synapse.lib.cell.CellApi method*), 658
getRoles() (*synapse.lib.hiveauth.HiveUser method*), 701
getRsrcTypes() (*in module synapse.lookup.pe*), 827
getRstText() (*synapse.lib.autodoc.RstHelp method*), 634
getRtypeStr() (*in module synapse.lib.autodoc*), 634
getRules() (*synapse.lib.hiveauth.HiveRuler method*), 700
getRuntNodes() (*synapse.lib.snap.Snap method*), 768
getRuntVars() (*synapse.lib.ast.AstNode method*), 618
getRuntVars() (*synapse.lib.ast.CatchBlock method*), 619
getRuntVars() (*synapse.lib.ast.EmbedQuery method*), 621
getRuntVars() (*synapse.lib.ast.ForLoop method*), 623
getRuntVars() (*synapse.lib.ast.Function method*), 624
getRuntVars() (*synapse.lib.ast.SetVarOper method*), 629
getRuntVars() (*synapse.lib.ast.VarListSetOper method*), 633
gets() (*synapse.lib.lmdbslab.MultiQueue method*), 731
gets() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
gets() (*synapse.lib.slabseqn.SlabSeqn method*), 765
getScopeVars() (*synapse.lib.storm.Runtime method*), 783
getSectionTypes() (*in module synapse.lookup.macho*), 827
getSeqn() (*synapse.lib.lmdbslab.Slab method*), 734
getServerSSLContext() (*in module synapse.lib.certdir*), 677
getServerSSLContext() (*synapse.lib.certdir.CertDir method*), 670
getSessInfo() (*synapse.daemon.Daemon method*), 925
getSessItem() (*synapse.daemon.Sess method*), 925
getSetOps() (*synapse.lib.snap.ProtoNode method*), 766
getShadow() (*in module synapse.lib.hiveauth*), 701
getShadowV2() (*in module synapse.lib.crypto.passwd*), 575
getShareInfo() (*in module synapse.lib.reflect*), 756
getSlabsInDir() (*synapse.lib.lmdbslab.Slab class method*), 734
getSlabStats() (*synapse.lib.lmdbslab.Slab class method*), 734
getSnapMeta() (*synapse.lib.snap.Snap method*), 768
getSpawnInfo() (*synapse.lib.link.Link method*), 728
getSpooledSet() (*synapse.lib.cell.Cell method*), 650
getsQueue() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getsQueue() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getSslCtx() (*in module synapse.common*), 889
getStatus() (*synapse.lib.health.HealthCheck method*), 693
getStemCell() (*in module synapse.servers.stemcell*), 838
getStorCmprs() (*synapse.lib.types.Type method*), 819

getStorIndx() (*synapse.lib.layer.Layer method*), 721
getStormCmd() (*synapse.cortex.Cortex method*), 910
getStormCmds() (*synapse.cortex.Cortex method*), 910
getStormCmds() (*synapse.lib.module.CoreModule method*), 739
getStormCmds() (*synapse.tests.utils.TestModule method*), 854
getStormDmon() (*synapse.cortex.CoreApi method*), 898
getStormDmon() (*synapse.cortex.Cortex method*), 910
getStormDmonLog() (*synapse.cortex.CoreApi method*), 898
getStormDmonLog() (*synapse.cortex.Cortex method*), 910
getStormDmons() (*synapse.cortex.CoreApi method*), 898
getStormDmons() (*synapse.cortex.Cortex method*), 910
getStormDocs() (*synapse.cortex.Cortex method*), 910
getStormGraph() (*synapse.cortex.Cortex method*), 910
getStormGraphs() (*synapse.cortex.Cortex method*), 910
getStormIfaces() (*synapse.cortex.Cortex method*), 910
getStormLib() (*synapse.cortex.Cortex method*), 910
getStormMacro() (*synapse.cortex.Cortex method*), 910
getStormMacros() (*synapse.cortex.Cortex method*), 910
getStormMod() (*synapse.cortex.Cortex method*), 910
getStormMods() (*synapse.cortex.Cortex method*), 910
getStormPkg() (*synapse.cortex.CoreApi method*), 898
getStormPkg() (*synapse.cortex.Cortex method*), 910
getStormPkgs() (*synapse.cortex.CoreApi method*), 898
getStormPkgs() (*synapse.cortex.Cortex method*), 910
getStormPool() (*synapse.cortex.Cortex method*), 910
getStormQuery() (*synapse.cortex.Cortex method*), 910
getStormQuery() (*synapse.lib.storm.Runtime method*), 783
getStormRuntime() (*synapse.cortex.Cortex method*), 910
getStormRuntime() (*synapse.lib.snap.Snap method*), 769
getStormStr() (*in module synapse.tools.genpkg*), 862
getStormSvc() (*synapse.cortex.Cortex method*), 911
getStormSvcInfo() (*synapse.lib.stormsvc.StormSvc method*), 790
getStormSvcPkgs() (*synapse.lib.stormsvc.StormSvc method*), 790
getStormSvcs() (*synapse.cortex.Cortex method*), 911
getStormVar() (*synapse.cortex.CoreApi method*), 898
getStormVar() (*synapse.cortex.Cortex method*), 911
getStorNode() (*synapse.datamodel.Form method*), 926
getStorNode() (*synapse.datamodel.Prop method*), 929
getStorNode() (*synapse.datamodel.TagProp method*), 929
getStorNode() (*synapse.lib.layer.Layer method*), 721
getStorNode() (*synapse.lib.storm.Cmd class method*), 772
getStorNode() (*synapse.lib.stormtypes.Layer method*), 791
getStorNode() (*synapse.lib.trigger.Trigger method*), 812
getStorNode() (*synapse.lib.types.Type method*), 819
getStorNode() (*synapse.tests.utils.TestRunt method*), 855
getStorNodeCount() (*synapse.lib.layer.Layer method*), 721
getStorNodes() (*synapse.lib.layer.Layer method*), 721
getStorNodes() (*synapse.lib.node.Node method*), 746
getStorNodes() (*synapse.lib.stormtypes.Layer method*), 791
getStorNodes() (*synapse.lib.view.View method*), 824
getStructuredAsyncLoggerStream() (*synapse.tests.utils.SynTest method*), 845
getSubRuntime() (*synapse.lib.storm.Runtime method*), 784
getSynDir() (*in module synapse.common*), 889
getSynPath() (*in module synapse.common*), 889
getSysctls() (*in module synapse.lib.platforms.linux*), 580
getSystemInfo() (*synapse.lib.cell.Cell method*), 650
getSystemInfo() (*synapse.lib.cell.CellApi method*), 658
getTag() (*synapse.lib.node.Node method*), 746
getTag() (*synapse.lib.snap.ProtoNode method*), 766
getTagCount() (*synapse.lib.layer.Layer method*), 721
getTagFilt() (*synapse.lib.layer.StorTypeTag static method*), 728
getTagGlobRegx() (*in module synapse.lib.cache*), 642
getTagModel() (*synapse.cortex.Cortex method*), 911
getTagNode() (*synapse.lib.snap.Snap method*), 769
getTagNorm() (*synapse.lib.snap.Snap method*), 769
getTagProp() (*synapse.datamodel.Model method*), 928
getTagProp() (*synapse.lib.node.Node method*), 746
getTagProp() (*synapse.lib.snap.ProtoNode method*), 766
getTagPropAbrv() (*synapse.lib.layer.Layer method*), 721
getTagPropCount() (*synapse.lib.layer.Layer method*), 721
getTagPropCount() (*synapse.lib.view.View method*), 824
getTagPropDef() (*synapse.datamodel.TagProp method*), 929
getTagProps() (*synapse.lib.layer.Layer method*), 721
getTagProps() (*synapse.lib.node.Node method*), 746
getTagPropValuCount() (*synapse.lib.layer.Layer method*), 721
getTagPrune() (*synapse.cortex.Cortex method*), 911
getTags() (*synapse.lib.node.Node method*), 746

getTeleApi() (*synapse.lib.cell.Cell method*), 651
getTeleApi() (*synapse.lib.hive.Hive method*), 695
getTeleApi() (*synapse.telepath.Aware method*), 939
getTeleProxy() (*synapse.lib.storm.Runtime method*), 784
getTempCoreCmdr() (*in module synapse.lib.jupyter*), 716
getTempCoreCmdrStormsvc() (*in module synapse.lib.jupyter*), 716
getTempCoreProx() (*in module synapse.lib.jupyter*), 717
getTempCoreStorm() (*in module synapse.lib.jupyter*), 717
getTempCoreStormStormsvc() (*in module synapse.lib.jupyter*), 717
getTempCortex() (*in module synapse.cortex*), 922
getTempDir() (*in module synapse.common*), 889
getTempDir() (*in module synapse.lib.platforms.common*), 579
getTempDir() (*synapse.lib.cell.Cell method*), 651
getTestAha() (*synapse.tests.utils.SynTest method*), 846
getTestAhaProv() (*synapse.tests.utils.SynTest method*), 846
getTestAxon() (*synapse.tests.utils.SynTest method*), 846
getTestCell() (*synapse.tests.utils.SynTest method*), 846
getTestCertDir() (*synapse.tests.utils.SynTest method*), 846
getTestConfDir() (*synapse.tests.utils.SynTest method*), 847
getTestCore() (*synapse.tests.utils.StormPkgTest method*), 840
getTestCore() (*synapse.tests.utils.SynTest method*), 847
getTestCoreAndProxy() (*synapse.tests.utils.SynTest method*), 847
getTestCoreProxSvc() (*synapse.tests.utils.SynTest method*), 847
getTestCryo() (*synapse.tests.utils.SynTest method*), 847
getTestCryoAndProxy() (*synapse.tests.utils.SynTest method*), 847
getTestDir() (*synapse.tests.utils.SynTest method*), 848
getTestDmon() (*synapse.tests.utils.SynTest method*), 848
getTestFilePath() (*synapse.tests.utils.SynTest method*), 848
getTestHive() (*synapse.tests.utils.SynTest method*), 848
getTestHiveDmon() (*synapse.tests.utils.SynTest method*), 848
getTestHiveFromDirn() (*synapse.tests.utils.SynTest method*), 848
getTestJsonStor() (*synapse.tests.utils.SynTest method*), 848
getTestOutp() (*synapse.tests.utils.SynTest method*), 848
getTestProxy() (*synapse.tests.utils.SynTest method*), 848
getTestReadWriteCores() (*synapse.tests.utils.SynTest method*), 848
getTestSynDir() (*synapse.tests.utils.SynTest method*), 849
getTestTeleHive() (*synapse.tests.utils.SynTest method*), 849
getTestUrl() (*synapse.tests.utils.SynTest method*), 849
getTickTock() (*synapse.lib.types.Time method*), 817
getTlsPeerCn() (*synapse.lib.link.Link method*), 728
getTotalMemory() (*in module synapse.lib.platforms.linux*), 580
getTrigger() (*synapse.lib.view.View method*), 824
getTypeClone() (*synapse.datamodel.Model method*), 928
getTypeDef() (*synapse.lib.types.Type method*), 819
getTypeDocs() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
getTypeNorm() (*synapse.cortex.CoreApi method*), 898
getTypeNorm() (*synapse.cortex.Cortex method*), 911
getTypeVals() (*synapse.lib.types.Type method*), 819
getTypeVals() (*synapse.models.inet.HttpCookie method*), 831
getTypeVals() (*synapse.models.inet.IPv4 method*), 831
getTypeVals() (*synapse.models.inet.IPv6 method*), 832
getTzNames() (*in module synapse.lookup.timezones*), 827
getTzOffset() (*in module synapse.lookup.timezones*), 827
getUnivPropCount() (*synapse.lib.layer.Layer method*), 721
getUserApiKey() (*synapse.lib.cell.Cell method*), 651
getUserByName() (*synapse.lib.hiveauth.Auth method*), 699
getUserCaPath() (*synapse.lib.certdir.CertDir method*), 670
getUserCert() (*synapse.lib.certdir.CertDir method*), 670
getUserCertPath() (*synapse.lib.certdir.CertDir method*), 671
getUserCsrPath() (*synapse.lib.certdir.CertDir method*), 671
getUserDef() (*synapse.lib.cell.Cell method*), 651
getUserDef() (*synapse.lib.cell.CellApi method*), 659
getUserDefByName() (*synapse.lib.cell.Cell method*), 651
getUserDefByName() (*synapse.lib.cell.CellApi method*), 659
getUserDefs() (*synapse.lib.cell.Cell method*), 651

getUserDefs() (*synapse.lib.cell.CellApi method*), 659
getUserForHost() (*synapse.lib.certdir.CertDir method*), 671
getUseridenBody() (*synapse.lib.httpapi.HandlerBase method*), 705
getUserIdenByName() (*synapse.lib.hiveauth.Auth method*), 699
getUserInfo() (*synapse.lib.aha.EnrollApi method*), 617
getUserInfo() (*synapse.lib.cell.CellApi method*), 659
getUserKey() (*synapse.lib.certdir.CertDir method*), 671
getUserKeyPath() (*synapse.lib.certdir.CertDir method*), 672
getUserName() (*synapse.lib.cell.Cell method*), 651
getUserNotif() (*synapse.cortex.CoreApi method*), 899
getUserNotif() (*synapse.cortex.Cortex method*), 911
getUserNotif() (*synapse.lib.jsonstor.JsonStorApi method*), 712
getUserNotif() (*synapse.lib.jsonstor.JsonStorCell method*), 713
getUserProfile() (*synapse.lib.cell.Cell method*), 651
getUserProfile() (*synapse.lib.cell.CellApi method*), 659
getUserProfInfo() (*synapse.lib.cell.Cell method*), 651
getUserProfInfo() (*synapse.lib.cell.CellApi method*), 659
getUserVarValu() (*synapse.lib.cell.Cell method*), 651
getVar() (*synapse.lib.node.Path method*), 748
getVar() (*synapse.lib.storm.Runtime method*), 784
getvars() (*in module synapse.lib.msgpack*), 741
getVault() (*synapse.cortex.Cortex method*), 911
getVaultByName() (*synapse.cortex.Cortex method*), 911
getVaultByType() (*synapse.cortex.Cortex method*), 911
getView() (*synapse.cortex.Cortex method*), 912
getViewDef() (*synapse.cortex.Cortex method*), 912
getViewDefs() (*synapse.cortex.Cortex method*), 912
getVolInfo() (*in module synapse.lib.platforms.common*), 579
GisLib (*class in synapse.lib.stormlib.gis*), 592
GovCnModule (*class in synapse.models.gov.cn*), 827
GovIntlModule (*class in synapse.models.gov.intl*), 828
GovUsModule (*class in synapse.models.gov.us*), 828
grant() (*synapse.lib.hiveauth.HiveUser method*), 701
GraphCmd (*class in synapse.lib.storm*), 775
GraphLib (*class in synapse.lib.stormlib.graph*), 592
gt() (*synapse.tests.utils.SynTest method*), 849
Guid (*class in synapse.lib.types*), 814
guid() (*in module synapse.common*), 890
guid() (*synapse.lib.hashset.HashSet method*), 693
GuidStor (*class in synapse.lib.lmdbslab*), 729
GzipLib (*class in synapse.lib.stormlib.compression*), 586

H

handleApiKeyAuth() (*synapse.lib.httpapi.HandlerBase method*), 705
handleBasicAuth() (*synapse.lib.httpapi.HandlerBase method*), 705
handleErr() (*synapse.lib.rstorm.StormCliOutput method*), 756
handleErr() (*synapse.tools.storm.StormCli method*), 866
handleList() (*in module synapse.tools.cellauth*), 861
handleModify() (*in module synapse.tools.cellauth*), 861
Handler (*class in synapse.lib.httpapi*), 704
HandlerBase (*class in synapse.lib.httpapi*), 704
handoff() (*synapse.lib.cell.Cell method*), 652
handoff() (*synapse.lib.cell.CellApi method*), 659
handshake() (*synapse.telepath.Proxy method*), 941
has() (*synapse.axon.Axon method*), 872
has() (*synapse.axon.AxonApi method*), 880
has() (*synapse.lib.lmdbslab.GuidStor method*), 729
has() (*synapse.lib.lmdbslab.Slab method*), 734
has() (*synapse.lib.node.Node method*), 746
has() (*synapse.lib.spooled.Dict method*), 770
has() (*synapse.lib.spooled.Set method*), 770
has() (*synapse.lib.stormlib.auth.UserJson method*), 584
has() (*synapse.lib.stormtypes.LibAxon method*), 792
has() (*synapse.lib.stormtypes.LibJsonStor method*), 795
has_dynamic_source_filename() (*synapse.utils.stormcov.plugin.PivotTracer method*), 868
has_dynamic_source_filename() (*synapse.utils.stormcov.plugin.StormCtrlTracer method*), 868
has_dynamic_source_filename() (*synapse.utils.stormcov.plugin.StormPlugin method*), 870
has_popts_data() (*in module synapse.tools.autodoc*), 859
HasAbsPropCond (*class in synapse.lib.ast*), 624
hasAstClass() (*synapse.lib.ast.AstNode method*), 618
hasChildTags() (*synapse.lib.storm.TagPruneCmd method*), 786
hasData() (*synapse.lib.node.Node method*), 746
hasdup() (*synapse.lib.lmdbslab.Slab method*), 734
hasglob() (*synapse.lib.ast.TagMatch method*), 631
hashes() (*synapse.axon.Axon method*), 873
hashes() (*synapse.axon.AxonApi method*), 880
hashitem() (*in module synapse.lib.hashitem*), 693
HashSet (*class in synapse.lib.hashset*), 693
hashset() (*synapse.axon.Axon method*), 873
hashset() (*synapse.axon.AxonApi method*), 880
hashset() (*synapse.lib.stormtypes.LibAxon method*), 792
hasHttpSess() (*synapse.lib.cell.Cell method*), 652

hasIndxBuid() (*synapse.lib.layer.IndxBy method*), 719
hasJsonObj() (*synapse.cortex.Cortex method*), 912
hasKids() (*synapse.lib.view.View method*), 824
hasNodeData() (*synapse.lib.layer.Layer method*), 721
hasNodeData() (*synapse.lib.snap.Snap method*), 769
hasNodeEdge() (*synapse.lib.layer.Layer method*), 721
hasNodeEdge() (*synapse.lib.snap.Snap method*), 769
hasPathObj() (*synapse.lib.jsonstor.JsonStor method*), 711
hasPathObj() (*synapse.lib.jsonstor.JsonStorApi method*), 712
hasPathObj() (*synapse.lib.jsonstor.JsonStorCell method*), 713
hasProp() (*synapse.lib.ast.HasRelPropCond method*), 624
HasRelPropCond (*class in synapse.lib.ast*), 624
hasRole() (*synapse.lib.hiveauth.HiveUser method*), 701
hasTag() (*synapse.lib.node.Node method*), 746
hasTagProp() (*synapse.lib.layer.Layer method*), 721
hasTagProp() (*synapse.lib.node.Node method*), 746
HasTagPropCond (*class in synapse.lib.ast*), 624
hasVarName() (*synapse.lib.ast.AstNode method*), 618
hasVarName() (*synapse.lib.ast.EmbedQuery method*), 621
hasVarName() (*synapse.lib.ast.VarValue method*), 633
haversine() (*in module synapse.lib.gis*), 691
head() (*synapse.axon.AxonHttpBySha256InvalidV1 method*), 885
head() (*synapse.axon.AxonHttpBySha256V1 method*), 885
head() (*synapse.lib.httpapi.ExtApiHandler method*), 703
head() (*synapse.tests.utils.HttpReflector method*), 840
HealthCheck (*class in synapse.lib.health*), 693
HealthCheckV1 (*class in synapse.lib.httpapi*), 707
help() (*synapse.lib.storm.Parser method*), 781
HelpCmd (*class in synapse.lib.storm*), 776
HelpCmd (*class in synapse.tools.storm*), 864
Hex (*class in synapse.lib.types*), 814
HexLib (*class in synapse.lib.stormlib.hex*), 593
hexstr() (*in module synapse.lib.chop*), 678
highlight_storm() (*in module synapse.lib.storm_format*), 788
Hist (*class in synapse.lib.lmdbslab*), 729
histfile (*synapse.lib.cli.Cli attribute*), 679
histfile (*synapse.tools.storm.StormCli attribute*), 866
history() (*synapse.axon.Axon method*), 873
history() (*synapse.axon.AxonApi method*), 880
HitLimit, 933
Hive (*class in synapse.lib.hive*), 694
HiveApi (*class in synapse.lib.hive*), 695
hiveapi (*synapse.cortex.Cortex attribute*), 912
HiveCmd (*class in synapse.cmds.hive*), 571
HiveDict (*class in synapse.lib.hive*), 695
hivepath (*synapse.lib.stormlib.model.LibModelEdge attribute*), 599
HiveRole (*class in synapse.lib.hiveauth*), 700
HiveRuler (*class in synapse.lib.hiveauth*), 700
HiveUser (*class in synapse.lib.hiveauth*), 700
holdHashLock() (*synapse.axon.Axon method*), 873
hostaddr() (*in module synapse.lib.thishost*), 810
HotCount (*class in synapse.lib.lmdbslab*), 729
HotKeyVal (*class in synapse.lib.lmdbslab*), 730
HOUR (*synapse.lib.agenda.TimeUnit attribute*), 612
hour() (*in module synapse.lib.time*), 811
hour() (*synapse.lib.stormtypes.LibTime method*), 799
htmlToText() (*in module synapse.lib.stormlib.mime*), 598
HttpApi (*class in synapse.lib.stormlib.cortex*), 587
HttpApiMethods (*class in synapse.lib.stormlib.cortex*), 588
HttpApiVars (*class in synapse.lib.stormlib.cortex*), 589
httpcodereason() (*in module synapse.common*), 890
HttpCookie (*class in synapse.models.inet*), 831
HttpHeaderDict (*class in synapse.lib.stormlib.cortex*), 589
HttpPermsList (*class in synapse.lib.stormlib.cortex*), 590
HttpReflector (*class in synapse.tests.utils*), 839
HttpReq (*class in synapse.lib.stormlib.cortex*), 590
HttpResp (*class in synapse.lib.stormhttp*), 789
hugeadd() (*in module synapse.common*), 890
hugediv() (*in module synapse.common*), 890
hugemod() (*in module synapse.common*), 890
hugemul() (*in module synapse.common*), 890
HugeNum (*class in synapse.lib.types*), 815
hugenum() (*in module synapse.common*), 890
hugepow() (*in module synapse.common*), 891
hugeround() (*in module synapse.common*), 891
hugescaleb() (*in module synapse.common*), 891
hugesub() (*in module synapse.common*), 891

I

iAmLoop() (*in module synapse.glob*), 938
iden() (*in module synapse.lib.node*), 748
iden() (*in module synapse.lib.threads*), 810
iden() (*synapse.cryotank.CryoTank method*), 923
iden() (*synapse.cryotank.TankApi method*), 925
iden() (*synapse.lib.crypto.ecc.PriKey method*), 573
iden() (*synapse.lib.crypto.ecc.PubKey method*), 574
iden() (*synapse.lib.crypto.rsa.PriKey method*), 576
iden() (*synapse.lib.crypto.rsa.PubKey method*), 577
iden() (*synapse.lib.node.Node method*), 746
iden() (*synapse.lib.snap.ProtoNode method*), 766
IdenCmd (*class in synapse.lib.storm*), 776
IfClause (*class in synapse.lib.ast*), 624
IfStmt (*class in synapse.lib.ast*), 624
ImapLib (*class in synapse.lib.stormlib.imap*), 593

ImapServer (*class in synapse.lib.stormlib.imap*), 593
Imei (*class in synapse.models.telco*), 837
imeicsum() (*in module synapse.models.telco*), 837
importFile() (*synapse.lib.certdir.CertDir method*), 672
Imsi (*class in synapse.models.telco*), 837
inc() (*synapse.lib.lmdbslab.HotCount method*), 730
inc() (*synapse.lib.lmdbslab.SlabDict method*), 736
inc() (*synapse.lib.stormlib.stats.StatTally method*), 606
InconsistentStorage, 933
incref() (*synapse.lib.base.Base method*), 636
index() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
index() (*synapse.lib.nexus.NexsRoot method*), 743
index() (*synapse.lib.slabseqn.SlabSeqn method*), 765
indx() (*synapse.lib.layer.StorType method*), 726
indx() (*synapse.lib.layer.StorTypeFloat method*), 726
indx() (*synapse.lib.layer.StorTypeFqdn method*), 726
indx() (*synapse.lib.layer.StorTypeGuid method*), 727
indx() (*synapse.lib.layer.StorTypeHier method*), 727
indx() (*synapse.lib.layer.StorTypeHugeNum method*), 727
indx() (*synapse.lib.layer.StorTypeInt method*), 727
indx() (*synapse.lib.layer.StorTypeIpv6 method*), 727
indx() (*synapse.lib.layer.StorTypeIval method*), 727
indx() (*synapse.lib.layer.StorTypeLatLon method*), 727
indx() (*synapse.lib.layer.StorTypeMsgp method*), 728
indx() (*synapse.lib.layer.StorTypeUtf8 method*), 728
IndxBy (*class in synapse.lib.layer*), 719
indxBy() (*synapse.lib.layer.StorType method*), 726
IndxByForm (*class in synapse.lib.layer*), 719
indxByForm() (*synapse.lib.layer.StorType method*), 726
IndxByProp (*class in synapse.lib.layer*), 720
indxByProp() (*synapse.lib.layer.StorType method*), 726
IndxByPropArray (*class in synapse.lib.layer*), 720
indxByPropArray() (*synapse.lib.layer.StorType method*), 726
IndxByTag (*class in synapse.lib.layer*), 720
IndxByTagProp (*class in synapse.lib.layer*), 720
indxByTagProp() (*synapse.lib.layer.StorType method*), 726
inet_ntop() (*in module synapse.lib.platforms.common*), 579
inet_pton() (*in module synapse.lib.platforms.common*), 579
inet_server_check() (*in module synapse.lib.scrape*), 762
inetHttpConnect() (*synapse.lib.stormhttp.LibHttp method*), 789
InetModule (*class in synapse.models.inet*), 832
info() (*synapse.cryotank.CryoTank method*), 924
ingest() (*synapse.lib.stormlib.stix.LibStixImport method*), 608
init() (*synapse.cryotank.CryoApi method*), 922
init() (*synapse.cryotank.CryoCell method*), 923
init() (*synapse.exc.NoSuchForm class method*), 935
init() (*synapse.exc.NoSuchProp class method*), 936
init() (*synapse.lib.ast.AstNode method*), 618
init2() (*synapse.lib.view.View method*), 824
InitBlock (*class in synapse.lib.ast*), 624
initCellApi() (*synapse.lib.cell.CellApi method*), 659
initCellConf() (*synapse.lib.cell.Cell class method*), 652
initCmdClasses() (*synapse.lib.cli.Cli method*), 679
initCmdClasses() (*synapse.tools.storm.StormCli method*), 866
initCoreModule() (*synapse.lib.module.CoreModule method*), 739
initCoreModule() (*synapse.models.files.FileModule method*), 829
initCoreModule() (*synapse.models.inet.InetModule method*), 832
initCoreModule() (*synapse.models.infotech.ItModule method*), 833
initCoreModule() (*synapse.models.proj.ProjectModule method*), 835
initCoreModule() (*synapse.models.syn.SynModule method*), 836
initCoreModule() (*synapse.tests.utils.TestModule method*), 854
initdb() (*synapse.lib.lmdbslab.Slab method*), 734
initframe() (*synapse.lib.node.Path method*), 748
initFromArgv() (*synapse.lib.cell.Cell class method*), 652
initHostInfo() (*in module synapse.lib.platforms.common*), 579
initHostInfo() (*in module synapse.lib.platforms.darwin*), 580
initHostInfo() (*in module synapse.lib.platforms.freebsd*), 580
initHostInfo() (*in module synapse.lib.platforms.linux*), 580
initHostInfo() (*in module synapse.lib.platforms.windows*), 581
initialize() (*synapse.lib.httpapi.HandlerBase method*), 706
initLayerActive() (*synapse.lib.layer.Layer method*), 721
initLayerPassive() (*synapse.lib.layer.Layer method*), 721
initLibAsync() (*synapse.lib.stormtypes.Lib method*), 792
initloop() (*in module synapse.glob*), 938
initMergeTask() (*synapse.lib.view.View method*), 824
initNexusSubsystem() (*synapse.lib.cell.Cell method*), 652
initPath() (*synapse.lib.storm.Runtime method*), 784
initPhoneTree() (*in module*

synapse.lookup.phonenum), 827
initServiceActive() (*synapse.cortex.Cortex method*), 912
initServiceActive() (*synapse.lib.cell.Cell method*), 653
initServiceNetwork() (*synapse.lib.aha.AhaCell method*), 616
initServiceNetwork() (*synapse.lib.cell.Cell method*), 653
initServicePassive() (*synapse.cortex.Cortex method*), 912
initServicePassive() (*synapse.lib.cell.Cell method*), 653
initServiceRuntime() (*synapse.axon.Axon method*), 873
initServiceRuntime() (*synapse.cortex.Cortex method*), 912
initServiceRuntime() (*synapse.lib.aha.AhaCell method*), 616
initServiceRuntime() (*synapse.lib.cell.Cell method*), 653
initServiceStorage() (*synapse.axon.Axon method*), 873
initServiceStorage() (*synapse.cortex.Cortex method*), 912
initServiceStorage() (*synapse.lib.aha.AhaCell method*), 616
initServiceStorage() (*synapse.lib.cell.Cell method*), 653
initServiceStorage() (*synapse.lib.jsonstor.JsonStorCell method*), 713
initSslCtx() (*synapse.lib.cell.Cell method*), 653
initStormPool() (*synapse.cortex.Cortex method*), 912
initSubRuntime() (*synapse.lib.storm.Runtime method*), 784
initSyncLoop() (*synapse.lib.lmdbslab.Slab class method*), 734
initTestCore() (*synapse.tests.utils.StormPkgTest method*), 841
initTrigTask() (*synapse.lib.view.View method*), 824
initUpstreamSync() (*synapse.lib.layer.Layer method*), 721
inline() (*synapse.lib.ast.SubQuery method*), 630
Int (*class in synapse.lib.types*), 815
int64en() (*in module synapse.common*), 891
int64un() (*in module synapse.common*), 891
IntBase (*class in synapse.lib.types*), 815
IntersectCmd (*class in synapse.lib.storm*), 777
intify() (*in module synapse.common*), 891
intify() (*in module synapse.lib.stormtypes*), 808
intstr() (*in module synapse.lib.chop*), 678
IPv4 (*class in synapse.models.inet*), 831
ipv4 (*synapse.lib.platforms.windows.sockaddr attribute*), 581
IPv4Range (*class in synapse.models.inet*), 832
IPv6 (*class in synapse.models.inet*), 832
ipv6 (*synapse.lib.platforms.windows.sockaddr attribute*), 581
ipv6_check() (*in module synapse.lib.scrape*), 762
IPv6Range (*class in synapse.models.inet*), 832
isActiveCoro() (*synapse.lib.cell.Cell method*), 653
isAdmin() (*synapse.lib.hiveauth.HiveUser method*), 701
isAdmin() (*synapse.lib.storm.Runtime method*), 784
isafork() (*synapse.lib.view.View method*), 824
isArchived() (*synapse.lib.hiveauth.HiveUser method*), 701
isarray (*synapse.lib.types.Array attribute*), 813
isarray (*synapse.lib.types.Type attribute*), 819
isatitem() (*synapse.lib.lmdbslab.Scan method*), 731
isatitem() (*synapse.lib.lmdbslab.ScanKeys method*), 732
isBasePropNoPivprop() (*in module synapse.lib.grammar*), 692
isbuidhex() (*in module synapse.common*), 891
isCaCert() (*synapse.lib.certdir.CertDir method*), 672
isCellActive() (*synapse.lib.cell.Cell method*), 653
isCellActive() (*synapse.lib.cell.CellApi method*), 659
isClientCert() (*synapse.lib.certdir.CertDir method*), 673
isCmdName() (*in module synapse.lib.grammar*), 692
isCodeCert() (*synapse.lib.certdir.CertDir method*), 673
iscoro() (*in module synapse.lib.coro*), 688
IsDeprLocked, 934
IsFini, 934
isForkOf() (*synapse.lib.view.View method*), 824
isFormName() (*in module synapse.lib.grammar*), 692
isguid() (*in module synapse.common*), 891
isHostCert() (*synapse.lib.certdir.CertDir method*), 673
isin() (*synapse.tests.utils.SynTest method*), 849
isinstance() (*synapse.tests.utils.SynTest method*), 849
isLocked() (*synapse.lib.hiveauth.HiveUser method*), 701
isMergeReady() (*synapse.lib.view.View method*), 824
ismutable() (*in module synapse.lib.stormtypes*), 808
ismutable() (*synapse.lib.stormtypes.StormType method*), 805
isNexsReady() (*synapse.lib.nexus.NexsRoot method*), 743
isok() (*in module synapse.lib.msgpack*), 741
isOrigHost() (*synapse.lib.httpapi.HandlerBase method*), 706
isPropName() (*in module synapse.lib.grammar*), 692
IsReadOnly, 934
isReadOnly() (*synapse.lib.storm.Cmd method*), 772
isRoleAllowed() (*synapse.lib.cell.Cell method*), 653

isRoleAllowed() (*synapse.lib.cell.CellApi method*), 659
IsRuntForm, 934
isRuntSafe() (*synapse.lib.ast.ArgvQuery method*), 618
isRuntSafe() (*synapse.lib.ast.AstNode method*), 618
isRuntSafe() (*synapse.lib.ast.Const method*), 619
isRuntSafe() (*synapse.lib.ast.Function method*), 624
isRuntSafe() (*synapse.lib.ast.PropValue method*), 628
isRuntSafe() (*synapse.lib.ast.SubQuery method*), 630
isRuntSafe() (*synapse.lib.ast.TagValue method*), 632
isRuntSafe() (*synapse.lib.ast.Value method*), 632
isRuntSafe() (*synapse.lib.ast.VarValue method*), 633
isRuntSafeAtom() (*synapse.lib.ast.AstNode method*), 618
isRuntSafeAtom() (*synapse.lib.ast.PropValue method*), 628
isRuntSafeAtom() (*synapse.lib.ast.TagValue method*), 632
isRuntSafeAtom() (*synapse.lib.ast.VarValue method*), 633
isRuntVar() (*synapse.lib.storm.Runtime method*), 784
issue() (*synapse.lib.cell.CellApi method*), 659
issue() (*synapse.lib.nexus.NexsRoot method*), 743
isTagValid() (*synapse.cortex.Cortex method*), 912
isterm (*synapse.lib.parser.AstInfo attribute*), 753
istufo() (*synapse.tests.utils.SynTest method*), 849
isUnivName() (*in module synapse.lib.grammar*), 692
isUserAdmin() (*synapse.lib.httpapi.HandlerBase method*), 706
isUserAllowed() (*synapse.lib.cell.Cell method*), 653
isUserAllowed() (*synapse.lib.cell.CellApi method*), 659
isUserCert() (*synapse.lib.certdir.CertDir method*), 674
items() (*synapse.exc.SynErr method*), 938
items() (*synapse.lib.base.BaseRef method*), 639
items() (*synapse.lib.cache.LruDict method*), 642
items() (*synapse.lib.hive.HiveDict method*), 696
items() (*synapse.lib.lmdbslab.SlabDict method*), 736
items() (*synapse.lib.spooled.Dict method*), 770
itemsStormVar() (*synapse.cortex.Cortex method*), 912
iter() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
iter() (*synapse.lib.nexus.NexsRoot method*), 743
iter() (*synapse.lib.scope.Scope method*), 758
iter() (*synapse.lib.slabseqn.SlabSeqn method*), 765
iter() (*synapse.lib.stormlib.auth.UserJson method*), 584
iter() (*synapse.lib.stormlib.auth.UserProfile method*), 584
iter() (*synapse.lib.stormlib.auth.UserVars method*), 584
iter() (*synapse.lib.stormlib.cortex.HttpApiMethods method*), 589
iter() (*synapse.lib.stormlib.project.LibProjects method*), 602
iter() (*synapse.lib.stormlib.project.ProjectEpics method*), 603
iter() (*synapse.lib.stormlib.project.ProjectSprints method*), 603
iter() (*synapse.lib.stormlib.project.ProjectTicketComments method*), 604
iter() (*synapse.lib.stormlib.project.ProjectTickets method*), 604
iter() (*synapse.lib.stormlib.spooled.SpooledSet method*), 605
iter() (*synapse.lib.stormlib.stats.StatTally method*), 606
iter() (*synapse.lib.stormlib.vault.VaultConfigs method*), 609
iter() (*synapse.lib.stormlib.xml.XmlElement method*), 610
iter() (*synapse.lib.stormtypes.CmdOpts method*), 790
iter() (*synapse.lib.stormtypes.Dict method*), 791
iter() (*synapse.lib.stormtypes.LibJsonStor method*), 795
iter() (*synapse.lib.stormtypes.List method*), 800
iter() (*synapse.lib.stormtypes.NodeProps method*), 801
iter() (*synapse.lib.stormtypes.PathMeta method*), 802
iter() (*synapse.lib.stormtypes.PathVars method*), 802
iter() (*synapse.lib.stormtypes.Prim method*), 802
iter() (*synapse.lib.stormtypes.Query method*), 804
iter() (*synapse.lib.stormtypes.Set method*), 804
iter() (*synapse.lib.stormtypes.StormHiveDict method*), 805
iterBack() (*synapse.lib.slabseqn.SlabSeqn method*), 765
iterBackupArchive() (*synapse.lib.cell.Cell method*), 653
iterBackupArchive() (*synapse.lib.cell.CellApi method*), 659
iterdata() (*in module synapse.lib.encoding*), 690
iterData() (*synapse.lib.node.Node method*), 746
iterDataKeys() (*synapse.lib.node.Node method*), 746
iterEdgeNodes() (*synapse.lib.storm.LiftByVerb method*), 777
iterEdgesN1() (*synapse.lib.node.Node method*), 746
iterEdgesN2() (*synapse.lib.node.Node method*), 746
iterEdgeVerbs() (*synapse.lib.layer.Layer method*), 721
iterEdgeVerbs() (*synapse.lib.node.Node method*), 746
iterEdgeVerbs() (*synapse.lib.snap.Snap method*), 769
iterfd() (*in module synapse.common*), 891
iterfd() (*in module synapse.lib.msgpack*), 741
iterfile() (*in module synapse.lib.msgpack*), 741
iterFormRows() (*synapse.cortex.CoreApi method*), 899
iterFormRows() (*synapse.cortex.Cortex method*), 913
iterFormRows() (*synapse.lib.layer.Layer method*), 722
iterFqdnUp() (*in module synapse.lib.certdir*), 677

iterfunc() (*synapse.lib.lmdbslab.Scan method*), 732
iterfunc() (*synapse.lib.lmdbslab.ScanBack method*), 732
iterfunc() (*synapse.lib.lmdbslab.ScanKeys method*), 732
iterLayerNodeEdits() (*synapse.lib.layer.Layer method*), 722
iterLayerNodeEdits() (*synapse.lib.layer.LayerApi method*), 725
iterLibs() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
iterMpkFile() (*synapse.axon.Axon method*), 873
iterMpkFile() (*synapse.axon.AxonApi method*), 881
iterNewBackupArchive() (*synapse.lib.cell.Cell method*), 653
iterNewBackupArchive() (*synapse.lib.cell.CellApi method*), 659
iternext() (*synapse.lib.lmdbslab.Scan method*), 732
iternext() (*synapse.lib.lmdbslab.ScanKeys method*), 732
iterNodeData() (*synapse.lib.layer.Layer method*), 722
iterNodeData() (*synapse.lib.snap.Snap method*), 769
iterNodeDataKeys() (*synapse.lib.layer.Layer method*), 722
iterNodeDataKeys() (*synapse.lib.snap.Snap method*), 769
iterNodeEdgesN1() (*synapse.lib.layer.Layer method*), 722
iterNodeEdgesN1() (*synapse.lib.snap.Snap method*), 769
iterNodeEdgesN2() (*synapse.lib.layer.Layer method*), 722
iterNodeEdgesN2() (*synapse.lib.snap.Snap method*), 769
iterNodeEditLog() (*synapse.lib.layer.Layer method*), 722
iterNodeEditLogBack() (*synapse.lib.layer.Layer method*), 722
iterNodePaths() (*synapse.lib.ast.Query method*), 628
iterpath() (*in module synapse.lib.hive*), 697
iterPoolTopo() (*synapse.lib.aha.AhaApi method*), 614
iterPoolTopo() (*synapse.lib.aha.AhaCell method*), 616
iterPropRows() (*synapse.cortex.CoreApi method*), 899
iterPropRows() (*synapse.cortex.Cortex method*), 913
iterPropRows() (*synapse.lib.layer.Layer method*), 722
iterright() (*synapse.lib.ast.AstNode method*), 618
iterStormPodes() (*synapse.lib.snap.Snap method*), 769
iterStormPodes() (*synapse.lib.view.View method*), 824
iterTagPropRows() (*synapse.cortex.CoreApi method*), 899
iterTagPropRows() (*synapse.cortex.Cortex method*), 913
iterTagPropRows() (*synapse.lib.layer.Layer method*), 722
iterTagRows() (*synapse.cortex.CoreApi method*), 900
iterTagRows() (*synapse.cortex.Cortex method*), 913
iterTagRows() (*synapse.lib.layer.Layer method*), 723
iterTypes() (*synapse.lib.stormtypes.StormTypesRegistry method*), 805
iterUnivRows() (*synapse.cortex.CoreApi method*), 900
iterUnivRows() (*synapse.cortex.Cortex method*), 914
iterUnivRows() (*synapse.lib.layer.Layer method*), 723
iterUserNotifs() (*synapse.cortex.CoreApi method*), 900
iterUserNotifs() (*synapse.cortex.Cortex method*), 914
iterUserNotifs() (*synapse.lib.jsonstor.JsonStorApi method*), 712
iterUserNotifs() (*synapse.lib.jsonstor.JsonStorCell method*), 713
iterUserVars() (*synapse.lib.cell.Cell method*), 653
iterWipeNodeEdits() (*synapse.lib.layer.Layer method*), 723
iterzip() (*in module synapse.common*), 891
ItModule (*class in synapse.models.infotech*), 833
Ival (*class in synapse.lib.types*), 815
ival() (*in module synapse.lib.time*), 811

J

join() (*synapse.lib.stormtypes.LibStr method*), 798
jslines() (*in module synapse.common*), 891
jsload() (*in module synapse.common*), 891
JsonFormatter (*class in synapse.lib.structlog*), 809
JsonLib (*class in synapse.lib.stormlib.json*), 596
jsonlines() (*synapse.axon.Axon method*), 873
jsonlines() (*synapse.axon.AxonApi method*), 881
jsonlines() (*synapse.lib.stormtypes.LibAxon method*), 792
jsonsafe_nodeedits() (*in module synapse.common*), 892
JsonSchema (*class in synapse.lib.stormlib.json*), 596
JsonStor (*class in synapse.lib.jsonstor*), 711
JsonStorApi (*class in synapse.lib.jsonstor*), 711
JsonStorCell (*class in synapse.lib.jsonstor*), 712
jssave() (*in module synapse.common*), 892

K

keepalive() (*synapse.lib.snap.Snap method*), 769
keyBuidsByDups() (*synapse.lib.layer.IndxBy method*), 719
keyBuidsByDupsBack() (*synapse.lib.layer.IndxBy method*), 719
keyBuidsByPref() (*synapse.lib.layer.IndxBy method*), 719

keyBuidsByPrefBack() (*synapse.lib.layer.IndxBy method*), 719
keyBuidsByRange() (*synapse.lib.layer.IndxBy method*), 719
keyBuidsByRangeBack() (*synapse.lib.layer.IndxBy method*), 719
keys() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
keys() (*synapse.lib.lmdbslab.SlabDict method*), 736
keys() (*synapse.lib.spooled.Dict method*), 770
kill() (*synapse.lib.cell.Cell method*), 653
kill() (*synapse.lib.cell.CellApi method*), 660
kill() (*synapse.lib.task.Task method*), 809
KillCmd (*class in synapse.cmds.boss*), 569
known_types (*synapse.lib.stormtypes.StormTypesRegistry attribute*), 805
kwarg_format() (*in module synapse.lib.stormtypes*), 808

L

LangModule (*class in synapse.models.language*), 834
last() (*synapse.cryotank.CryoApi method*), 922
last() (*synapse.cryotank.CryoTank method*), 924
last() (*synapse.lib.lmdbslab.Slab method*), 734
last() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
last() (*synapse.lib.slabseqn.SlabSeqn method*), 765
lastkey() (*synapse.lib.lmdbslab.Slab method*), 734
LatLong (*class in synapse.models.geospace*), 830
latlong() (*in module synapse.lib.gis*), 691
Layer (*class in synapse.lib.layer*), 720
Layer (*class in synapse.lib.stormtypes*), 791
LayerApi (*class in synapse.lib.layer*), 725
layerapi (*synapse.cortex.Cortex attribute*), 914
layerConfirm() (*synapse.lib.storm.Runtime method*), 784
LayerInUse, 934
layrctor() (*synapse.cortex.Cortex class method*), 914
le() (*synapse.tests.utils.SynTest method*), 849
leave() (*synapse.lib.scope.Scope method*), 758
len() (*synapse.tests.utils.SynTest method*), 849
Lib (*class in synapse.lib.stormtypes*), 792
LibAuth (*class in synapse.lib.stormlib.auth*), 582
LibAxon (*class in synapse.lib.stormtypes*), 792
LibBase (*class in synapse.lib.stormtypes*), 793
LibBase64 (*class in synapse.lib.stormtypes*), 793
LibBytes (*class in synapse.lib.stormtypes*), 793
LibCache (*class in synapse.lib.stormlib.cache*), 585
LibCron (*class in synapse.lib.stormtypes*), 793
LibCsv (*class in synapse.lib.stormtypes*), 794
LibDict (*class in synapse.lib.stormtypes*), 794
LibDmon (*class in synapse.lib.stormtypes*), 794
LibEasyPerm (*class in synapse.lib.stormlib.easyperm*), 591
LibExport (*class in synapse.lib.stormtypes*), 794
LibFeed (*class in synapse.lib.stormtypes*), 795
LibGates (*class in synapse.lib.stormlib.auth*), 582
LibGen (*class in synapse.lib.stormlib.gen*), 591
LibGlobals (*class in synapse.lib.stormtypes*), 795
LibHashes (*class in synapse.lib.stormlib.hashes*), 592
LibHmac (*class in synapse.lib.stormlib.hashes*), 592
LibHttp (*class in synapse.lib.stormhttp*), 789
LibIpv6 (*class in synapse.lib.stormlib.ipv6*), 595
LibIters (*class in synapse.lib.stormlib.iters*), 595
LibJsonStor (*class in synapse.lib.stormtypes*), 795
LibLayer (*class in synapse.lib.stormtypes*), 795
LibLift (*class in synapse.lib.stormtypes*), 796
LibMacro (*class in synapse.lib.stormlib.macro*), 597
LibMimeHtml (*class in synapse.lib.stormlib.mime*), 598
LibModel (*class in synapse.lib.stormlib.model*), 598
LibModelDeprecated (*class in synapse.lib.stormlib.model*), 598
LibModelEdge (*class in synapse.lib.stormlib.model*), 598
LibModelExt (*class in synapse.lib.stormlib.modelext*), 600
LibModelTags (*class in synapse.lib.stormlib.model*), 599
LibPack (*class in synapse.lib.stormlib.pack*), 601
LibPipe (*class in synapse.lib.stormtypes*), 796
LibPkg (*class in synapse.lib.stormtypes*), 796
LibProjects (*class in synapse.lib.stormlib.project*), 602
LibPs (*class in synapse.lib.stormtypes*), 796
LibQueue (*class in synapse.lib.stormtypes*), 797
LibRandom (*class in synapse.lib.stormlib.random*), 604
LibRegx (*class in synapse.lib.stormtypes*), 797
LibRoles (*class in synapse.lib.stormlib.auth*), 582
LibScrape (*class in synapse.lib.stormlib.scrape*), 604
LibService (*class in synapse.lib.stormtypes*), 797
LibSpooled (*class in synapse.lib.stormlib.spooled*), 605
LibStats (*class in synapse.lib.stormlib.stats*), 606
LibStix (*class in synapse.lib.stormlib.stix*), 607
LibStixExport (*class in synapse.lib.stormlib.stix*), 607
LibStixImport (*class in synapse.lib.stormlib.stix*), 607
LibStorm (*class in synapse.lib.stormlib.storm*), 608
LibStr (*class in synapse.lib.stormtypes*), 797
LibTags (*class in synapse.lib.stormtypes*), 798
LibTelepath (*class in synapse.lib.stormtypes*), 798
LibTime (*class in synapse.lib.stormtypes*), 798
LibTrigger (*class in synapse.lib.stormtypes*), 799
LibTst (*class in synapse.tests.utils*), 840
LibUser (*class in synapse.lib.stormlib.auth*), 582
LibUsers (*class in synapse.lib.stormlib.auth*), 583
LibVars (*class in synapse.lib.stormtypes*), 799
LibVault (*class in synapse.lib.stormlib.vault*), 609
LibView (*class in synapse.lib.stormtypes*), 799
LibWhois (*class in synapse.lib.stormwhois*), 809
LibXml (*class in synapse.lib.stormlib.xml*), 610
LibYaml (*class in synapse.lib.stormlib.yaml*), 610
lift() (*synapse.lib.ast.LiftByArray method*), 625
lift() (*synapse.lib.ast.LiftFormTag method*), 625

lift() (*synapse.lib.ast.LiftFormTagProp method*), 625
lift() (*synapse.lib.ast.LiftOper method*), 625
lift() (*synapse.lib.ast.LiftProp method*), 625
lift() (*synapse.lib.ast.LiftPropBy method*), 625
lift() (*synapse.lib.ast.LiftTag method*), 625
lift() (*synapse.lib.ast.LiftTagProp method*), 626
lift() (*synapse.lib.ast.LiftTagTag method*), 626
liftBundle() (*synapse.lib.stormlib.stix.LibStix method*), 607
LiftByArray (*class in synapse.lib.ast*), 625
liftByDataName() (*synapse.lib.layer.Layer method*), 723
liftByFormValu() (*synapse.lib.layer.Layer method*), 723
liftByProp() (*synapse.lib.layer.Layer method*), 723
liftByProp() (*synapse.lib.stormtypes.Layer method*), 791
liftByPropArray() (*synapse.lib.layer.Layer method*), 723
liftByPropValu() (*synapse.lib.layer.Layer method*), 723
liftByTag() (*synapse.lib.layer.Layer method*), 723
liftByTag() (*synapse.lib.stormtypes.Layer method*), 791
liftByTagProp() (*synapse.lib.layer.Layer method*), 723
liftByTagPropValu() (*synapse.lib.layer.Layer method*), 723
liftByTagValu() (*synapse.lib.layer.Layer method*), 723
LiftByVerb (*class in synapse.lib.storm*), 777
LiftFormTag (*class in synapse.lib.ast*), 625
LiftFormTagProp (*class in synapse.lib.ast*), 625
LiftOper (*class in synapse.lib.ast*), 625
LiftProp (*class in synapse.lib.ast*), 625
LiftPropBy (*class in synapse.lib.ast*), 625
liftreverse() (*synapse.lib.parser.AstConverter method*), 753
LiftTag (*class in synapse.lib.ast*), 625
LiftTagProp (*class in synapse.lib.ast*), 625
liftTagProp() (*synapse.lib.layer.Layer method*), 723
LiftTagTag (*class in synapse.lib.ast*), 626
LimitCmd (*class in synapse.lib.storm*), 777
line_number_range() (*synapse.utils.stormcov.plugin.PivotTracer method*), 868
line_number_range() (*synapse.utils.stormcov.plugin.StormCtrlTracer method*), 869
line_number_range() (*synapse.utils.stormcov.plugin.StormPlugin method*), 870
lines() (*synapse.utils.stormcov.plugin.StormReporter method*), 870
Link (*class in synapse.lib.link*), 728
link() (*synapse.lib.base.Base method*), 636
LinkBadCert, 934
LinkErr, 934
linkfile() (*in module synapse.lib.link*), 729
LinkShutDown, 934
linksock() (*in module synapse.lib.link*), 729
linux_path_check() (*in module synapse.lib.scrape*), 762
List (*class in synapse.lib.ast*), 626
List (*class in synapse.lib.stormtypes*), 800
list() (*synapse.cryotank.CryoApi method*), 922
list() (*synapse.cryotank.CryoCell method*), 923
list() (*synapse.lib.agenda.Agenda method*), 612
list() (*synapse.lib.coro.GenrHelp method*), 687
list() (*synapse.lib.lmdbslab.MultiQueue method*), 731
list() (*synapse.lib.stormlib.imap.ImapServer method*), 594
list() (*synapse.lib.stormlib.notifications.NotifyLib method*), 600
list() (*synapse.lib.stormtypes.LibAxon method*), 792
list() (*synapse.lib.stormtypes.NodeProps method*), 801
list() (*synapse.lib.trigger.Triggers method*), 812
list() (*synapse.telepath.GenrIter method*), 940
listCoreQueues() (*synapse.cortex.Cortex method*), 914
listCronJobs() (*synapse.cortex.Cortex method*), 914
listdir() (*in module synapse.common*), 892
listen() (*in module synapse.lib.link*), 729
listen() (*synapse.daemon.Daemon method*), 925
listHiveKey() (*synapse.lib.cell.Cell method*), 653
listHiveKey() (*synapse.lib.cell.CellApi method*), 660
listHttpApis() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
listLayers() (*synapse.cortex.Cortex method*), 914
listOAuthClients() (*synapse.lib.oauth.OAuthMixin method*), 751
listOAuthProviders() (*synapse.lib.oauth.OAuthMixin method*), 752
listTagModel() (*synapse.cortex.Cortex method*), 914
listTriggers() (*synapse.lib.view.View method*), 824
listUserApiKeys() (*synapse.lib.cell.Cell method*), 653
listVaults() (*synapse.cortex.Cortex method*), 914
listViews() (*synapse.cortex.Cortex method*), 914
ljuster() (*in module synapse.lib.autodoc*), 634
LmdbBackup (*class in synapse.lib.lmdbslab*), 730
LmdbLock, 934
load() (*synapse.lib.crypto.ecc.PriKey static method*), 573
load() (*synapse.lib.crypto.ecc.PubKey static method*), 574
load() (*synapse.lib.crypto.rsa.PubKey static method*), 577

load() (*synapse.lib.stormlib.yaml.LibYaml method*), 611
load() (*synapse.lib.trigger.Triggers method*), 812
load() (*synapse.tools.storm.StormCompleter method*), 867
loadCertByts() (*synapse.lib.certdir.CertDir method*), 674
loadCoreModule() (*synapse.cortex.Cortex method*), 914
loadfile() (*in module synapse.lib.msgpack*), 741
loadHiveTree() (*synapse.lib.cell.Cell method*), 653
loadHiveTree() (*synapse.lib.hive.Hive method*), 695
loadHiveTree() (*synapse.lib.hive.HiveApi method*), 695
loadJsonMesg() (*synapse.lib.httpapi.HandlerBase method*), 706
loadNode() (*synapse.lib.snap.SnapEditor method*), 770
loadOpticFiles() (*in module synapse.tools.genpkg*), 862
loadOpticWorkflows() (*in module synapse.tools.genpkg*), 862
loadPkgProto() (*in module synapse.tools.genpkg*), 862
loadStormPkg() (*synapse.cortex.Cortex method*), 915
loadTeleCell() (*in module synapse.telepath*), 942
loadTeleEnv() (*in module synapse.telepath*), 942
Loc (*class in synapse.lib.types*), 816
localSchemaRefHandler() (*in module synapse.lib.config*), 686
Log (*class in synapse.cmds.cortex*), 570
logAuthIssue() (*synapse.lib.httpapi.HandlerBase method*), 706
logger (*in module synapse.lib.crypto.coin*), 572
LoggerLib (*class in synapse.lib.stormlib.log*), 596
login() (*synapse.lib.httpapi.Sess method*), 708
login() (*synapse.lib.stormlib.imap.ImapServer method*), 594
LoginV1 (*class in synapse.lib.httpapi*), 707
logout() (*synapse.lib.httpapi.Sess method*), 708
LogoutV1 (*class in synapse.lib.httpapi*), 707
LookList (*class in synapse.lib.ast*), 626
Lookup (*class in synapse.lib.ast*), 626
lookup() (*synapse.lib.parser.Parser method*), 754
lookupedgesforform() (*in module synapse.tools.autodoc*), 859
loop() (*in module synapse.lib.task*), 809
LowSpace, 934
LruDict (*class in synapse.lib.cache*), 641
lt() (*synapse.tests.utils.SynTest method*), 849

M

MacroExecCmd (*class in synapse.lib.stormlib.macro*), 597
main() (*in module synapse.lib.base*), 640
main() (*in module synapse.servers.cell*), 838
main() (*in module synapse.servers.stemcell*), 838
main() (*in module synapse.tools.aha.easycert*), 857
main() (*in module synapse.tools.aha.enroll*), 857
main() (*in module synapse.tools.aha.list*), 857
main() (*in module synapse.tools.aha.provision.service*), 857
main() (*in module synapse.tools.aha.provision.user*), 857
main() (*in module synapse.tools.autodoc*), 859
main() (*in module synapse.tools.axon2axon*), 860
main() (*in module synapse.tools.backup*), 860
main() (*in module synapse.tools.cellauth*), 861
main() (*in module synapse.tools.cmdr*), 861
main() (*in module synapse.tools.cryo.cat*), 857
main() (*in module synapse.tools.cryo.list*), 858
main() (*in module synapse.tools.csvtool*), 861
main() (*in module synapse.tools.docker.validate*), 858
main() (*in module synapse.tools.easycert*), 861
main() (*in module synapse.tools.feed*), 861
main() (*in module synapse.tools.genpkg*), 862
main() (*in module synapse.tools.guid*), 862
main() (*in module synapse.tools.healthcheck*), 863
main() (*in module synapse.tools.hive.load*), 858
main() (*in module synapse.tools.hive.save*), 858
main() (*in module synapse.tools.json2mpk*), 863
main() (*in module synapse.tools.livebackup*), 863
main() (*in module synapse.tools.modrole*), 863
main() (*in module synapse.tools.moduser*), 863
main() (*in module synapse.tools.promote*), 863
main() (*in module synapse.tools.pullfile*), 863
main() (*in module synapse.tools.pushfile*), 864
main() (*in module synapse.tools.reload*), 864
main() (*in module synapse.tools.rstorm*), 864
main() (*in module synapse.tools.storm*), 867
main() (*in module synapse.utils.getrefs*), 871
main() (*synapse.lib.base.Base method*), 636
make_envar_name() (*in module synapse.lib.config*), 686
makeargparser() (*in module synapse.tools.autodoc*), 859
makeargparser() (*in module synapse.tools.cellauth*), 861
makeargparser() (*in module synapse.tools.csvtool*), 861
makeargparser() (*in module synapse.tools.feed*), 861
makeargparser() (*in module synapse.tools.healthcheck*), 863
makeargparser() (*in module synapse.tools.pushfile*), 864
makeColLook() (*in module synapse.lookup.iso3166*), 826
makedirs() (*in module synapse.common*), 892
makeHttpResponse() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
markSeen() (*synapse.lib.stormlib.imap.ImapServer method*), 594

massage_vartokn() (*in module synapse.lib.parser*), 754
matchContexts() (*in module synapse.tests.utils*), 856
matches() (*in module synapse.lib.version*), 821
matches() (*synapse.lib.stormlib.version.VersionLib method*), 610
matches() (*synapse.lib.stormtypes.LibRegx method*), 797
MathLib (*class in synapse.lib.stormlib.math*), 597
MatModule (*class in synapse.models.material*), 835
MaxCmd (*class in synapse.lib.storm*), 778
maximizeMaxLockedMemory() (*in module synapse.lib.platforms.linux*), 580
mayDelBuid() (*synapse.lib.layer.Layer method*), 724
MediaModule (*class in synapse.models.media*), 835
meh() (*in module synapse.lib.grammar*), 692
memoize() (*in module synapse.lib.cache*), 642
memoizemethod() (*in module synapse.lib.cache*), 642
merge() (*synapse.lib.types.Int method*), 815
merge() (*synapse.lib.types.Ival method*), 815
merge() (*synapse.lib.types.Time method*), 817
merge() (*synapse.lib.types.Type method*), 819
merge() (*synapse.lib.view.View method*), 825
mergeAhaInfo() (*in module synapse.telepath*), 942
mergeAllowed() (*synapse.lib.view.View method*), 825
MergeCmd (*class in synapse.lib.storm*), 778
mergeStormIface() (*synapse.lib.view.View method*), 825
merggenr() (*in module synapse.common*), 892
merggenr2() (*in module synapse.common*), 892
message() (*synapse.lib.stormlib.smtp.SmtpLib method*), 605
meta() (*synapse.lib.node.Path method*), 748
metaToAstInfo() (*synapse.lib.parser.AstConverter method*), 753
Method (*class in synapse.telepath*), 940
metrics() (*synapse.axon.Axon method*), 874
metrics() (*synapse.axon.AxonApi method*), 881
metrics() (*synapse.cryotank.CryoApi method*), 922
metrics() (*synapse.cryotank.CryoTank method*), 924
metrics() (*synapse.cryotank.TankApi method*), 925
metrics() (*synapse.lib.stormtypes.LibAxon method*), 792
MinCmd (*class in synapse.lib.storm*), 779
MINUTE (*synapse.lib.agenda.TimeUnit attribute*), 612
minute() (*in module synapse.lib.time*), 811
minute() (*synapse.lib.stormtypes.LibTime method*), 799
MitreAttackFlowLib (*class in synapse.lib.stormlib.infosec*), 595
mlock() (*in module synapse.lib.platforms.linux*), 580
mmap() (*in module synapse.lib.platforms.linux*), 580
mod() (*synapse.lib.agenda.Agenda method*), 612
mod_name (*synapse.lib.module.CoreModule attribute*), 739
modAhaSvcInfo() (*synapse.lib.aha.AhaApi method*), 614
modAhaSvcInfo() (*synapse.lib.aha.AhaCell method*), 616
ModAlreadyLoaded, 934
modCellConf() (*synapse.lib.cell.Cell method*), 653
Model (*class in synapse.datamodel*), 927
ModelForm (*class in synapse.lib.stormlib.model*), 599
ModelNormV1 (*class in synapse.lib.httpapi*), 707
ModelProp (*class in synapse.lib.stormlib.model*), 599
ModelRev (*class in synapse.lib.modelrev*), 737
ModelTagProp (*class in synapse.lib.stormlib.model*), 599
ModelType (*class in synapse.lib.stormlib.model*), 599
ModelV1 (*class in synapse.lib.httpapi*), 708
modHttpExtApi() (*synapse.cortex.Cortex method*), 915
modStormGraph() (*synapse.cortex.Cortex method*), 915
modStormMacro() (*synapse.cortex.Cortex method*), 915
module
    synapse, 569
    synapse.axon, 871
    synapse.cells, 886
    synapse.cmds, 569
    synapse.cmds.boss, 569
    synapse.cmds.cortex, 570
    synapse.cmds.hive, 571
    synapse.common, 886
    synapse.cortex, 895
    synapse.cryotank, 922
    synapse.daemon, 925
    synapse.data, 571
    synapse.datamodel, 926
    synapse.exc, 930
    synapse.glob, 938
    synapse.lib, 572
    synapse.lib.agenda, 611
    synapse.lib.aha, 613
    synapse.lib.ast, 617
    synapse.lib.autodoc, 634
    synapse.lib.base, 635
    synapse.lib.boss, 641
    synapse.lib.cache, 641
    synapse.lib.cell, 642
    synapse.lib.certdir, 662
    synapse.lib.chop, 677
    synapse.lib.cli, 679
    synapse.lib.cmd, 682
    synapse.lib.cmdr, 682
    synapse.lib.config, 683
    synapse.lib.const, 687
    synapse.lib.coro, 687
    synapse.lib.crypto, 572
    synapse.lib.crypto.coin, 572
    synapse.lib.crypto.ecc, 572
    synapse.lib.crypto.passwd, 575

synapse.lib.crypto.rsa, 576
synapse.lib.crypto.tinfoil, 578
synapse.lib.datfile, 689
synapse.lib.dyndeps, 689
synapse.lib.encoding, 690
synapse.lib.gis, 691
synapse.lib.grammar, 692
synapse.lib.hashitem, 693
synapse.lib.hashset, 693
synapse.lib.health, 693
synapse.lib.hive, 694
synapse.lib.hiveauth, 697
synapse.lib.httpapi, 702
synapse.lib.ingest, 710
synapse.lib.interval, 710
synapse.lib.jsonstor, 711
synapse.lib.jupyter, 713
synapse.lib.layer, 718
synapse.lib.link, 728
synapse.lib.lmdbslab, 729
synapse.lib.modelrev, 737
synapse.lib.module, 738
synapse.lib.modules, 740
synapse.lib.msgpack, 740
synapse.lib.multislabseqn, 742
synapse.lib.nexus, 743
synapse.lib.node, 745
synapse.lib.oauth, 751
synapse.lib.output, 752
synapse.lib.parser, 752
synapse.lib.platforms, 579
synapse.lib.platforms.common, 579
synapse.lib.platforms.darwin, 580
synapse.lib.platforms.freebsd, 580
synapse.lib.platforms.linux, 580
synapse.lib.platforms.windows, 581
synapse.lib.queue, 754
synapse.lib.ratelimit, 755
synapse.lib.reflect, 755
synapse.lib.rstorm, 756
synapse.lib.schemas, 757
synapse.lib.scope, 757
synapse.lib.scrape, 759
synapse.lib.share, 764
synapse.lib.slaboffs, 764
synapse.lib.slabseqn, 764
synapse.lib.snap, 766
synapse.lib.spooled, 770
synapse.lib.storm, 771
synapse.lib.storm_format, 788
synapse.lib.stormctrl, 788
synapse.lib.stormhttp, 789
synapse.lib.stormlib, 581
synapse.lib.stormlib.aha, 581
synapse.lib.stormlib.auth, 582
synapse.lib.stormlib.backup, 585
synapse.lib.stormlib.basex, 585
synapse.lib.stormlib.cache, 585
synapse.lib.stormlib.cell, 586
synapse.lib.stormlib.compression, 586
synapse.lib.stormlib.cortex, 587
synapse.lib.stormlib.easyperm, 591
synapse.lib.stormlib.ethereum, 591
synapse.lib.stormlib.gen, 591
synapse.lib.stormlib.gis, 592
synapse.lib.stormlib.graph, 592
synapse.lib.stormlib.hashes, 592
synapse.lib.stormlib.hex, 593
synapse.lib.stormlib.imap, 593
synapse.lib.stormlib.infosec, 594
synapse.lib.stormlib.ipv6, 595
synapse.lib.stormlib.iters, 595
synapse.lib.stormlib.json, 596
synapse.lib.stormlib.log, 596
synapse.lib.stormlib.macro, 597
synapse.lib.stormlib.math, 597
synapse.lib.stormlib.mime, 598
synapse.lib.stormlib.model, 598
synapse.lib.stormlib.modelext, 600
synapse.lib.stormlib.notifications, 600
synapse.lib.stormlib.oauth, 601
synapse.lib.stormlib.pack, 601
synapse.lib.stormlib.project, 602
synapse.lib.stormlib.random, 604
synapse.lib.stormlib.scrape, 604
synapse.lib.stormlib.smtp, 605
synapse.lib.stormlib.spooled, 605
synapse.lib.stormlib.stats, 606
synapse.lib.stormlib.stix, 607
synapse.lib.stormlib.storm, 608
synapse.lib.stormlib.vault, 609
synapse.lib.stormlib.version, 610
synapse.lib.stormlib.xml, 610
synapse.lib.stormlib.yaml, 610
synapse.lib.stormsvc, 790
synapse.lib.stormtypes, 790
synapse.lib.stormwhois, 809
synapse.lib.structlog, 809
synapse.lib.task, 809
synapse.lib.thishost, 810
synapse.lib.thisplat, 810
synapse.lib.threads, 810
synapse.lib.time, 811
synapse.lib.trigger, 812
synapse.lib.types, 813
synapse.lib.urlhelp, 820
synapse.lib.version, 821
synapse.lib.view, 823

synapse.lookup, 826
synapse.lookup.cvss, 826
synapse.lookup.iana, 826
synapse.lookup.iso3166, 826
synapse.lookup.macho, 827
synapse.lookup.pe, 827
synapse.lookup.phonenum, 827
synapse.lookup.timezones, 827
synapse.mindmeld, 939
synapse.models, 827
synapse.models.auth, 828
synapse.models.base, 828
synapse.models.belief, 828
synapse.models.biz, 829
synapse.models.crypto, 829
synapse.models.dns, 829
synapse.models.economic, 829
synapse.models.files, 829
synapse.models.geopol, 830
synapse.models.geospace, 830
synapse.models.gov, 827
synapse.models.gov.cn, 827
synapse.models.gov.intl, 828
synapse.models.gov.us, 828
synapse.models.inet, 831
synapse.models.infotech, 833
synapse.models.language, 834
synapse.models.material, 835
synapse.models.media, 835
synapse.models.orgs, 835
synapse.models.person, 835
synapse.models.proj, 835
synapse.models.risk, 836
synapse.models.science, 836
synapse.models.syn, 836
synapse.models.telco, 837
synapse.models.transport, 838
synapse.servers, 838
synapse.servers.aha, 838
synapse.servers.axon, 838
synapse.servers.cell, 838
synapse.servers.cortex, 838
synapse.servers.cryotank, 838
synapse.servers.jsonstor, 838
synapse.servers.stemcell, 838
synapse.telepath, 939
synapse.tests, 838
synapse.tests.nopmod, 838
synapse.tests.utils, 839
synapse.tools, 857
synapse.tools.aha, 857
synapse.tools.aha.easycert, 857
synapse.tools.aha.enroll, 857
synapse.tools.aha.list, 857
synapse.tools.aha.provision, 857
synapse.tools.aha.provision.service, 857
synapse.tools.aha.provision.user, 857
synapse.tools.autodoc, 858
synapse.tools.axon2axon, 860
synapse.tools.backup, 860
synapse.tools.cellauth, 861
synapse.tools.cmdr, 861
synapse.tools.cryo, 857
synapse.tools.cryo.cat, 857
synapse.tools.cryo.list, 858
synapse.tools.csvtool, 861
synapse.tools.docker, 858
synapse.tools.docker.validate, 858
synapse.tools.easycert, 861
synapse.tools.feed, 861
synapse.tools.genpkg, 862
synapse.tools.guid, 862
synapse.tools.healthcheck, 863
synapse.tools.hive, 858
synapse.tools.hive.load, 858
synapse.tools.hive.save, 858
synapse.tools.json2mpk, 863
synapse.tools.livebackup, 863
synapse.tools.modrole, 863
synapse.tools.moduser, 863
synapse.tools.promote, 863
synapse.tools.pullfile, 863
synapse.tools.pushfile, 864
synapse.tools.reload, 864
synapse.tools.rstorm, 864
synapse.tools.storm, 864
synapse.utils, 867
synapse.utils.getrefs, 871
synapse.utils.stormcov, 867
synapse.utils.stormcov.plugin, 868
modurl() (*in module synapse.telepath*), 942
modUserApiKey() (*synapse.lib.cell.Cell method*), 654
mononow() (*in module synapse.common*), 892
MONTH (*synapse.lib.agenda.TimeUnit attribute*), 612
month() (*in module synapse.lib.time*), 811
month() (*synapse.lib.stormtypes.LibTime method*), 799
monthofyear() (*synapse.lib.stormtypes.LibTime method*), 799
move() (*synapse.lib.agenda.Agenda method*), 612
move() (*synapse.lib.stormtypes.Trigger method*), 807
moveCronJob() (*synapse.cortex.Cortex method*), 915
MoveNodesCmd (*class in synapse.lib.storm*), 779
MoveTagCmd (*class in synapse.lib.storm*), 780
MultiQueue (*class in synapse.lib.lmdbslab*), 731
MultiSlabSeqn (*class in synapse.lib.multislabseqn*), 742
munlock() (*in module synapse.lib.platforms.linux*), 580
MustBeJsonSafe, 934

N

N1Walk (*class in synapse.lib.ast*), 626
N1WalkNPivo (*class in synapse.lib.ast*), 626
N2Walk (*class in synapse.lib.ast*), 626
N2WalkNPivo (*class in synapse.lib.ast*), 626
name (*synapse.lib.storm.BackgroundCmd attribute*), 771
name (*synapse.lib.storm.BatchCmd attribute*), 771
name (*synapse.lib.storm.Cmd attribute*), 772
name (*synapse.lib.storm.CopyToCmd attribute*), 773
name (*synapse.lib.storm.CountCmd attribute*), 773
name (*synapse.lib.storm.DelNodeCmd attribute*), 773
name (*synapse.lib.storm.DiffCmd attribute*), 774
name (*synapse.lib.storm.DivertCmd attribute*), 774
name (*synapse.lib.storm.EdgesDelCmd attribute*), 775
name (*synapse.lib.storm.GraphCmd attribute*), 775
name (*synapse.lib.storm.HelpCmd attribute*), 776
name (*synapse.lib.storm.IdenCmd attribute*), 776
name (*synapse.lib.storm.IntersectCmd attribute*), 777
name (*synapse.lib.storm.LiftByVerb attribute*), 777
name (*synapse.lib.storm.LimitCmd attribute*), 778
name (*synapse.lib.storm.MaxCmd attribute*), 778
name (*synapse.lib.storm.MergeCmd attribute*), 779
name (*synapse.lib.storm.MinCmd attribute*), 779
name (*synapse.lib.storm.MoveNodesCmd attribute*), 780
name (*synapse.lib.storm.MoveTagCmd attribute*), 780
name (*synapse.lib.storm.OnceCmd attribute*), 781
name (*synapse.lib.storm.ParallelCmd attribute*), 781
name (*synapse.lib.storm.ReIndexCmd attribute*), 782
name (*synapse.lib.storm.RunAsCmd attribute*), 782
name (*synapse.lib.storm.ScrapeCmd attribute*), 785
name (*synapse.lib.storm.SleepCmd attribute*), 785
name (*synapse.lib.storm.SpinCmd attribute*), 785
name (*synapse.lib.storm.TagPruneCmd attribute*), 786
name (*synapse.lib.storm.TeeCmd attribute*), 786
name (*synapse.lib.storm.TreeCmd attribute*), 787
name (*synapse.lib.storm.UniqCmd attribute*), 787
name (*synapse.lib.storm.ViewExecCmd attribute*), 788
name (*synapse.lib.stormlib.cortex.StormPoolDelCmd attribute*), 590
name (*synapse.lib.stormlib.cortex.StormPoolGetCmd attribute*), 590
name (*synapse.lib.stormlib.cortex.StormPoolSetCmd attribute*), 591
name (*synapse.lib.stormlib.macro.MacroExecCmd attribute*), 597
name (*synapse.lib.stormlib.stats.StatsCountByCmd attribute*), 607
name (*synapse.tests.utils.TestCmd attribute*), 854
name() (*synapse.lib.hive.Node method*), 696
names() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
nameToAbrv() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
Ndef (*class in synapse.lib.types*), 816
ndef() (*in module synapse.lib.node*), 748
ne() (*synapse.tests.utils.SynTest method*), 849
near() (*in module synapse.lib.gis*), 692
NeedConfValu, 934
newkey() (*in module synapse.lib.crypto.tinfoil*), 579
NexsRoot (*class in synapse.lib.nexus*), 743
nextindx() (*synapse.lib.slabseqn.SlabSeqn method*), 765
nextitem() (*synapse.lib.storm.ParallelCmd method*), 781
nexttime() (*synapse.lib.agenda.ApptRec method*), 612
nn() (*synapse.tests.utils.SynTest method*), 849
NoCertKey, 934
Node (*class in synapse.lib.hive*), 696
Node (*class in synapse.lib.node*), 745
Node (*class in synapse.lib.stormtypes*), 800
NodeData (*class in synapse.lib.stormtypes*), 800
nodeeditctor (*synapse.lib.layer.Layer attribute*), 724
NodeProp (*class in synapse.lib.types*), 816
NodeProps (*class in synapse.lib.stormtypes*), 800
nodes() (*synapse.cortex.Cortex method*), 915
nodes() (*synapse.lib.snap.Snap method*), 769
nodes() (*synapse.lib.stormlib.project.Project method*), 602
nodes() (*synapse.lib.stormlib.project.ProjectEpic method*), 602
nodes() (*synapse.lib.stormlib.project.ProjectSprint method*), 603
nodes() (*synapse.lib.stormlib.project.ProjectTicket method*), 603
nodes() (*synapse.lib.stormlib.project.ProjectTicketComment method*), 603
nodes() (*synapse.lib.stormtypes.Prim method*), 802
nodes() (*synapse.lib.stormtypes.Query method*), 804
nodes() (*synapse.lib.view.View method*), 825
nodesByDataName() (*synapse.lib.snap.Snap method*), 769
nodesByProp() (*synapse.lib.snap.Snap method*), 769
nodesByPropArray() (*synapse.lib.snap.Snap method*), 769
nodesByPropTypeValu() (*synapse.lib.snap.Snap method*), 769
nodesByPropValu() (*synapse.lib.snap.Snap method*), 769
nodesByTag() (*synapse.lib.snap.Snap method*), 769
nodesByTagProp() (*synapse.lib.snap.Snap method*), 769
nodesByTagPropValu() (*synapse.lib.snap.Snap method*), 769
nodesByTagValu() (*synapse.lib.snap.Snap method*), 769
nom() (*in module synapse.lib.grammar*), 692
none() (*synapse.tests.utils.SynTest method*), 849
noprop() (*synapse.tests.utils.SynTest method*), 849
norm() (*in module synapse.tests.utils*), 856

norm() (*synapse.lib.types.Data method*), 813
norm() (*synapse.lib.types.HugeNum method*), 815
norm() (*synapse.lib.types.Type method*), 819
norm() (*synapse.tests.utils.TestSubType method*), 855
norm() (*synapse.tests.utils.ThreeType method*), 855
normdict() (*in module synapse.lib.hashitem*), 693
normitem() (*in module synapse.lib.hashitem*), 693
normiter() (*in module synapse.lib.hashitem*), 693
normLogLevel() (*in module synapse.common*), 892
normOAuthTokenData() (*in module synapse.lib.oauth*), 752
NoSuchAbrv, 934
NoSuchAct, 934
NoSuchAuthGate, 934
NoSuchCert, 934
NoSuchCmd, 935
NoSuchCmpr, 935
NoSuchCond, 935
NoSuchCtor, 935
NoSuchDecoder, 935
NoSuchDir, 935
NoSuchDyn, 935
NoSuchEncoder, 935
NoSuchFile, 935
NoSuchForm, 935
NoSuchFunc, 935
NoSuchIden, 935
NoSuchImpl, 935
NoSuchIndx, 935
NoSuchLayer, 935
NoSuchLift, 935
NoSuchMeth, 935
NoSuchName, 935
NoSuchObj, 935
NoSuchOpt, 936
NoSuchPath, 936
NoSuchPivot, 936
NoSuchPkg, 936
NoSuchProp, 936
NoSuchRole, 936
NoSuchStormSvc, 936
NoSuchTagProp, 936
NoSuchType, 936
NoSuchUniv, 936
NoSuchUser, 936
NoSuchVar, 936
NoSuchView, 936
NotANumberCompared, 936
NotCond (*class in synapse.lib.ast*), 626
NotifyLib (*class in synapse.lib.stormlib.notifications*), 600
notin() (*synapse.tests.utils.SynTest method*), 850
NotMsgpackSafe, 936
NotReady, 936
NoValu (*class in synapse.common*), 886
NOW (*synapse.lib.agenda.TimeUnit attribute*), 612
now() (*in module synapse.common*), 892
Number (*class in synapse.lib.stormtypes*), 801

O

OAuthMixin (*class in synapse.lib.oauth*), 751
OAuthV1Client (*class in synapse.lib.stormlib.oauth*), 601
OAuthV1Lib (*class in synapse.lib.stormlib.oauth*), 601
OAuthV2Lib (*class in synapse.lib.stormlib.oauth*), 601
off() (*synapse.lib.base.Base method*), 636
offAdd() (*synapse.datamodel.Form method*), 926
offlink() (*synapse.telepath.Client method*), 939
offset() (*synapse.lib.lmdbslab.MultiQueue method*), 731
oflight (*synapse.lib.types.Velocity attribute*), 820
omit() (*synapse.lib.ast.SubGraph method*), 630
on() (*synapse.lib.base.Base method*), 637
on_connection_close() (*synapse.axon.AxonHttpUploadV1 method*), 885
on_connection_close() (*synapse.lib.httpapi.Handler method*), 704
on_finish() (*synapse.axon.AxonHttpUploadV1 method*), 885
on_message() (*synapse.lib.httpapi.BeholdSockV1 method*), 703
onAdd() (*synapse.datamodel.Form method*), 926
OnceCmd (*class in synapse.lib.storm*), 780
onDel() (*synapse.datamodel.Form method*), 926
onDel() (*synapse.datamodel.Prop method*), 929
OnePassIssueV1 (*class in synapse.lib.httpapi*), 708
onespace() (*in module synapse.lib.chop*), 678
onfini() (*synapse.lib.base.Base method*), 637
onInitMessage() (*synapse.lib.httpapi.BeholdSockV1 method*), 703
onlink() (*synapse.telepath.Client method*), 939
onPush() (*synapse.lib.nexus.Pusher class method*), 744
onPushAuto() (*synapse.lib.nexus.Pusher class method*), 744
onSet() (*synapse.datamodel.Prop method*), 929
onStormMesg() (*synapse.cmds.cortex.Log method*), 570
onTeleShare() (*synapse.telepath.Aware method*), 939
onWith() (*synapse.lib.base.Base method*), 637
open() (*in module synapse.telepath*), 942
open() (*synapse.lib.hive.Hive method*), 695
open() (*synapse.lib.hive.Node method*), 696
open() (*synapse.lib.hive.TeleHive method*), 697
openDatFile() (*in module synapse.lib.datfile*), 689
opendir() (*in module synapse.lib.hive*), 697
openinfo() (*in module synapse.telepath*), 942
openLogFd() (*synapse.cmds.cortex.Log method*), 570
openurl() (*in module synapse.lib.hive*), 697

Oper (*class in synapse.lib.ast*), 627
operrelprop_join() (*synapse.lib.parser.AstConverter method*), 753
operrelprop_pivot() (*synapse.lib.parser.AstConverter method*), 753
optimize() (*synapse.lib.ast.AstNode method*), 618
options() (*synapse.lib.httpapi.ExtApiHandler method*), 704
options() (*synapse.lib.httpapi.HandlerBase method*), 706
OrCond (*class in synapse.lib.ast*), 627
ornot() (*in module synapse.lib.coro*), 688
OuModule (*class in synapse.models.orgs*), 835
OutPut (*class in synapse.lib.output*), 752
OutPutBytes (*class in synapse.lib.output*), 752
OutPutFd (*class in synapse.lib.output*), 752
OutPutRst (*class in synapse.lib.rstorm*), 756
OutPutStr (*class in synapse.lib.output*), 752
overlap() (*in module synapse.lib.interval*), 710

P

pack() (*synapse.daemon.Sess method*), 925
pack() (*synapse.datamodel.Edge method*), 926
pack() (*synapse.datamodel.Form method*), 926
pack() (*synapse.datamodel.Prop method*), 929
pack() (*synapse.datamodel.TagProp method*), 929
pack() (*synapse.lib.agenda.ApptRec method*), 612
pack() (*synapse.lib.health.HealthCheck method*), 693
pack() (*synapse.lib.hive.HiveDict method*), 696
pack() (*synapse.lib.hiveauth.AuthGate method*), 700
pack() (*synapse.lib.hiveauth.HiveRole method*), 700
pack() (*synapse.lib.hiveauth.HiveUser method*), 701
pack() (*synapse.lib.layer.Layer method*), 724
pack() (*synapse.lib.lmdbslab.HotKeyVal method*), 730
pack() (*synapse.lib.node.Node method*), 746
pack() (*synapse.lib.node.Path method*), 748
pack() (*synapse.lib.storm.StormDmon method*), 785
pack() (*synapse.lib.stormlib.stix.StixBundle method*), 608
pack() (*synapse.lib.stormtypes.Trigger method*), 807
pack() (*synapse.lib.task.Task method*), 809
pack() (*synapse.lib.trigger.Trigger method*), 812
pack() (*synapse.lib.types.Type method*), 819
pack() (*synapse.lib.view.View method*), 825
packVersion() (*in module synapse.lib.version*), 821
ParallelCmd (*class in synapse.lib.storm*), 781
parent() (*synapse.lib.hive.Node method*), 696
parse() (*in module synapse.lib.time*), 811
parse() (*synapse.lib.stormlib.xml.LibXml method*), 610
parse_args() (*in module synapse.tools.backup*), 860
parse_args() (*in module synapse.utils.getrefs*), 871
parse_args() (*synapse.lib.storm.Parser method*), 781
parse_cmd_string() (*in module synapse.lib.parser*), 754
parse_float() (*in module synapse.lib.grammar*), 692
PARSE_METHODS (*synapse.utils.stormcov.plugin.PivotTracer attribute*), 868
PARSE_METHODS (*synapse.utils.stormcov.plugin.StormPlugin attribute*), 869
parseApiKey() (*in module synapse.lib.crypto.passwd*), 576
parseEval() (*in module synapse.lib.parser*), 754
parseNumber() (*in module synapse.lib.ast*), 634
parsepath() (*synapse.cmds.hive.HiveCmd static method*), 571
parseQuery() (*in module synapse.lib.parser*), 754
Parser (*class in synapse.lib.cmd*), 682
Parser (*class in synapse.lib.parser*), 753
Parser (*class in synapse.lib.storm*), 781
ParserExit, 936
parseSemver() (*in module synapse.lib.version*), 821
parsetime() (*in module synapse.lib.interval*), 710
parsetz() (*in module synapse.lib.time*), 811
parseVersionParts() (*in module synapse.lib.version*), 821
patch() (*synapse.lib.httpapi.ExtApiHandler method*), 704
Path (*class in synapse.lib.node*), 747
Path (*class in synapse.lib.stormtypes*), 801
path() (*in module synapse.data*), 572
PathExists, 936
PathMeta (*class in synapse.lib.stormtypes*), 802
PathVars (*class in synapse.lib.stormtypes*), 802
phnode() (*in module synapse.lookup.phonenum*), 827
Phone (*class in synapse.models.telco*), 837
PickleableMagicMock (*class in synapse.tests.utils*), 840
Pipe (*class in synapse.lib.stormtypes*), 802
Pipeline (*class in synapse.telepath*), 940
pipeline() (*synapse.lib.storm.ParallelCmd method*), 781
pipeline() (*synapse.lib.storm.TeeCmd method*), 786
pivogenr() (*synapse.lib.ast.FormPivot method*), 623
pivogenr() (*synapse.lib.ast.PropPivot method*), 628
PivotIn (*class in synapse.lib.ast*), 627
PivotInFrom (*class in synapse.lib.ast*), 627
PivotOper (*class in synapse.lib.ast*), 627
PivotOut (*class in synapse.lib.ast*), 627
pivots() (*synapse.lib.ast.SubGraph method*), 630
PivotToTags (*class in synapse.lib.ast*), 627
PivotTracer (*class in synapse.utils.stormcov.plugin*), 868
pkgname (*synapse.lib.storm.Cmd attribute*), 772
pkgprotos (*synapse.tests.utils.StormPkgTest attribute*), 841
PolModule (*class in synapse.models.geopol*), 830

pop() (*in module synapse.lib.scope*), 759
pop() (*synapse.lib.base.BaseRef method*), 640
pop() (*synapse.lib.cache.FixedCache method*), 641
pop() (*synapse.lib.hive.Hive method*), 695
pop() (*synapse.lib.hive.HiveDict method*), 696
pop() (*synapse.lib.hive.Node method*), 696
pop() (*synapse.lib.hive.TeleHive method*), 697
pop() (*synapse.lib.lmdbslab.MultiQueue method*), 731
pop() (*synapse.lib.lmdbslab.Slab method*), 734
pop() (*synapse.lib.lmdbslab.SlabDict method*), 736
pop() (*synapse.lib.node.Node method*), 747
pop() (*synapse.lib.scope.Scope method*), 758
pop() (*synapse.lib.slabseqn.SlabSeqn method*), 765
pop() (*synapse.lib.spooled.Dict method*), 770
pop() (*synapse.lib.trigger.Triggers method*), 812
popAndSync() (*synapse.lib.hive.HiveApi method*), 695
popCellConf() (*synapse.lib.cell.Cell method*), 654
popData() (*synapse.lib.node.Node method*), 747
popDmon() (*synapse.lib.storm.DmonManager method*), 775
popHiveKey() (*synapse.lib.cell.Cell method*), 654
popHiveKey() (*synapse.lib.cell.CellApi method*), 660
popPathObjProp() (*synapse.lib.jsonstor.JsonStor method*), 711
popPathObjProp() (*synapse.lib.jsonstor.JsonStorApi method*), 712
popPathObjProp() (*synapse.lib.jsonstor.JsonStorCell method*), 713
popSessItem() (*synapse.daemon.Sess method*), 925
popStormVar() (*synapse.cortex.CoreApi method*), 900
popStormVar() (*synapse.cortex.Cortex method*), 915
popTagModel() (*synapse.cortex.Cortex method*), 915
popUserProfInfo() (*synapse.lib.cell.Cell method*), 654
popUserProfInfo() (*synapse.lib.cell.CellApi method*), 660
popUserVarValu() (*synapse.lib.cell.Cell method*), 654
popVar() (*synapse.lib.node.Path method*), 748
popVar() (*synapse.lib.storm.Runtime method*), 784
post() (*synapse.axon.AxonHttpDelV1 method*), 885
post() (*synapse.axon.AxonHttpUploadV1 method*), 885
post() (*synapse.lib.aha.AhaProvisionServiceV1 method*), 617
post() (*synapse.lib.httpapi.AuthAddRoleV1 method*), 702
post() (*synapse.lib.httpapi.AuthAddUserV1 method*), 702
post() (*synapse.lib.httpapi.AuthDelRoleV1 method*), 702
post() (*synapse.lib.httpapi.AuthGrantV1 method*), 702
post() (*synapse.lib.httpapi.AuthRevokeV1 method*), 702
post() (*synapse.lib.httpapi.AuthRoleV1 method*), 702
post() (*synapse.lib.httpapi.AuthUserPasswdV1 method*), 703
post() (*synapse.lib.httpapi.AuthUserV1 method*), 703
post() (*synapse.lib.httpapi.ExtApiHandler method*), 704
post() (*synapse.lib.httpapi.FeedV1 method*), 704
post() (*synapse.lib.httpapi.LoginV1 method*), 707
post() (*synapse.lib.httpapi.ModelNormV1 method*), 708
post() (*synapse.lib.httpapi.OnePassIssueV1 method*), 708
post() (*synapse.lib.httpapi.ReqValidStormV1 method*), 708
post() (*synapse.lib.httpapi.StormCallV1 method*), 708
post() (*synapse.lib.httpapi.StormExportV1 method*), 709
post() (*synapse.lib.httpapi.StormNodesV1 method*), 709
post() (*synapse.lib.httpapi.StormV1 method*), 709
post() (*synapse.lib.httpapi.StormVarsPopV1 method*), 709
post() (*synapse.lib.httpapi.StormVarsSetV1 method*), 709
post() (*synapse.tests.utils.HttpReflector method*), 840
postAnit() (*synapse.lib.base.Base method*), 637
postAnit() (*synapse.tests.utils.ReloadCell method*), 840
postfiles() (*synapse.axon.Axon method*), 874
postfiles() (*synapse.axon.AxonApi method*), 881
postTypeInit() (*synapse.lib.types.Array method*), 813
postTypeInit() (*synapse.lib.types.Bool method*), 813
postTypeInit() (*synapse.lib.types.Comp method*), 813
postTypeInit() (*synapse.lib.types.Data method*), 814
postTypeInit() (*synapse.lib.types.Duration method*), 814
postTypeInit() (*synapse.lib.types.Edge method*), 814
postTypeInit() (*synapse.lib.types.Float method*), 814
postTypeInit() (*synapse.lib.types.Guid method*), 814
postTypeInit() (*synapse.lib.types.Hex method*), 814
postTypeInit() (*synapse.lib.types.Int method*), 815
postTypeInit() (*synapse.lib.types.Ival method*), 816
postTypeInit() (*synapse.lib.types.Loc method*), 816
postTypeInit() (*synapse.lib.types.Ndef method*), 816
postTypeInit() (*synapse.lib.types.NodeProp method*), 816
postTypeInit() (*synapse.lib.types.Range method*), 816
postTypeInit() (*synapse.lib.types.Str method*), 817
postTypeInit() (*synapse.lib.types.Tag method*), 817
postTypeInit() (*synapse.lib.types.TagPart method*), 817
postTypeInit() (*synapse.lib.types.Taxon method*), 817
postTypeInit() (*synapse.lib.types.Taxonomy method*), 817
postTypeInit() (*synapse.lib.types.Time method*), 818
postTypeInit() (*synapse.lib.types.TimeEdge method*), 818
postTypeInit() (*synapse.lib.types.Type method*), 819
postTypeInit() (*synapse.lib.types.Velocity method*), 820

postTypeInit() (*synapse.models.dns.DnsName method*), 829
postTypeInit() (*synapse.models.files.FileBase method*), 829
postTypeInit() (*synapse.models.files.FileBytes method*), 829
postTypeInit() (*synapse.models.files.FilePath method*), 830
postTypeInit() (*synapse.models.geospace.Area method*), 830
postTypeInit() (*synapse.models.geospace.Dist method*), 830
postTypeInit() (*synapse.models.geospace.LatLong method*), 830
postTypeInit() (*synapse.models.inet.Addr method*), 831
postTypeInit() (*synapse.models.inet.Cidr4 method*), 831
postTypeInit() (*synapse.models.inet.Cidr6 method*), 831
postTypeInit() (*synapse.models.inet.Email method*), 831
postTypeInit() (*synapse.models.inet.Fqdn method*), 831
postTypeInit() (*synapse.models.inet.IPv4 method*), 831
postTypeInit() (*synapse.models.inet.IPv4Range method*), 832
postTypeInit() (*synapse.models.inet.IPv6 method*), 832
postTypeInit() (*synapse.models.inet.IPv6Range method*), 832
postTypeInit() (*synapse.models.inet.Rfc2822Addr method*), 832
postTypeInit() (*synapse.models.inet.Url method*), 833
postTypeInit() (*synapse.models.infotech.SemVer method*), 834
postTypeInit() (*synapse.models.telco.Imei method*), 837
postTypeInit() (*synapse.models.telco.Imsi method*), 837
postTypeInit() (*synapse.models.telco.Phone method*), 837
postTypeInit() (*synapse.tests.utils.TestType method*), 855
preCoreModule() (*synapse.lib.module.CoreModule method*), 739
prefexists() (*synapse.lib.lmdbslab.Slab method*), 734
prefix (*synapse.lib.rstorm.OutPutRst attribute*), 756
prefix() (*synapse.lib.stormtypes.LibTags method*), 798
prepare() (*synapse.axon.AxonHttpUploadV1 method*), 885
prepare() (*synapse.cortex.CortexAxonMixin method*), 922
prepare() (*synapse.lib.ast.AstNode method*), 618
prepare() (*synapse.lib.ast.EditNodeAdd method*), 620
prepare() (*synapse.lib.ast.ExprDict method*), 622
prepare() (*synapse.lib.ast.ExprList method*), 622
prepare() (*synapse.lib.ast.ExprNode method*), 622
prepare() (*synapse.lib.ast.FormatString method*), 623
prepare() (*synapse.lib.ast.Function method*), 624
prepare() (*synapse.lib.ast.IfStmt method*), 624
prepare() (*synapse.lib.ast.List method*), 626
prepare() (*synapse.lib.ast.PropName method*), 628
prepare() (*synapse.lib.ast.PropValue method*), 628
prepare() (*synapse.lib.ast.SwitchCase method*), 630
prepare() (*synapse.lib.ast.TagName method*), 631
prepare() (*synapse.lib.ast.UnaryExprNode method*), 632
prepare() (*synapse.lib.ast.VarValue method*), 633
prepare() (*synapse.lib.httpapi.Handler method*), 704
prepareRstLines() (*in module synapse.lib.autodoc*), 634
PriKey (*class in synapse.lib.crypto.ecc*), 572
PriKey (*class in synapse.lib.crypto.rsa*), 576
Prim (*class in synapse.lib.stormtypes*), 802
printables() (*in module synapse.lib.chop*), 678
printed() (*synapse.tests.utils.SynTest method*), 850
printf() (*synapse.cmds.cortex.StormCmd method*), 571
printf() (*synapse.lib.cli.Cli method*), 680
printf() (*synapse.lib.cli.Cmd method*), 681
printf() (*synapse.lib.output.OutPut method*), 752
printf() (*synapse.lib.rstorm.OutPutRst method*), 756
printf() (*synapse.lib.rstorm.StormCliOutput method*), 756
printf() (*synapse.lib.rstorm.StormOutput method*), 757
printf() (*synapse.lib.snap.Snap method*), 769
printf() (*synapse.lib.storm.Runtime method*), 784
printf() (*synapse.tools.storm.StormCli method*), 866
printrole() (*in module synapse.tools.modrole*), 863
printuser() (*in module synapse.tools.cellauth*), 861
printuser() (*in module synapse.tools.moduser*), 863
processCtors() (*in module synapse.tools.autodoc*), 859
processFormsProps() (*in module synapse.tools.autodoc*), 859
processStormCmds() (*in module synapse.tools.autodoc*), 859
processStormModules() (*in module synapse.tools.autodoc*), 859
processTypes() (*in module synapse.tools.autodoc*), 859
processUnivs() (*in module synapse.tools.autodoc*), 859
Project (*class in synapse.lib.stormlib.project*), 602
ProjectEpic (*class in synapse.lib.stormlib.project*), 602
ProjectEpics (*class in synapse.lib.stormlib.project*), 602

ProjectModule (*class in synapse.models.proj*), 835
ProjectSprint (*class in synapse.lib.stormlib.project*), 603
ProjectSprints (*class in synapse.lib.stormlib.project*), 603
ProjectTicket (*class in synapse.lib.stormlib.project*), 603
ProjectTicketComment (*class in synapse.lib.stormlib.project*), 603
ProjectTicketComments (*class in synapse.lib.stormlib.project*), 603
ProjectTickets (*class in synapse.lib.stormlib.project*), 604
promote() (*synapse.lib.boss.Boss method*), 641
promote() (*synapse.lib.cell.Cell method*), 654
promote() (*synapse.lib.cell.CellApi method*), 660
promote() (*synapse.lib.nexus.NexsRoot method*), 743
promotetask() (*synapse.lib.boss.Boss method*), 641
prompt() (*synapse.lib.cli.Cli method*), 680
Prop (*class in synapse.datamodel*), 928
prop() (*in module synapse.lib.node*), 748
prop() (*synapse.datamodel.Form method*), 926
prop() (*synapse.datamodel.Model method*), 928
proplift() (*synapse.lib.ast.LiftProp method*), 625
PropName (*class in synapse.lib.ast*), 627
PropPivot (*class in synapse.lib.ast*), 628
PropPivotOut (*class in synapse.lib.ast*), 628
props() (*in module synapse.lib.node*), 748
PropValue (*class in synapse.lib.ast*), 628
ProtoNode (*class in synapse.lib.snap*), 766
ProvApi (*class in synapse.lib.aha*), 617
ProvDmon (*class in synapse.lib.aha*), 617
Proxy (*class in synapse.lib.stormtypes*), 802
Proxy (*class in synapse.telepath*), 940
proxy() (*synapse.telepath.Client method*), 939
proxy() (*synapse.telepath.ClientV2 method*), 940
ProxyGenrMethod (*class in synapse.lib.stormtypes*), 803
ProxyMethod (*class in synapse.lib.stormtypes*), 803
ps() (*synapse.lib.boss.Boss method*), 641
ps() (*synapse.lib.cell.Cell method*), 654
ps() (*synapse.lib.cell.CellApi method*), 660
PsCmd (*class in synapse.cmds.boss*), 569
PsModule (*class in synapse.models.person*), 835
PubKey (*class in synapse.lib.crypto.ecc*), 574
PubKey (*class in synapse.lib.crypto.rsa*), 576
public() (*synapse.lib.crypto.ecc.PriKey method*), 573
public() (*synapse.lib.crypto.rsa.PriKey method*), 576
PullFileCmd (*class in synapse.tools.storm*), 865
pullone() (*in module synapse.lib.ast*), 634
PureCmd (*class in synapse.lib.storm*), 781
Pusher (*class in synapse.lib.nexus*), 744
PushFileCmd (*class in synapse.tools.storm*), 865
put() (*synapse.axon.Axon method*), 875
put() (*synapse.axon.AxonApi method*), 881
put() (*synapse.axon.AxonHttpUploadV1 method*), 886
put() (*synapse.lib.base.BaseRef method*), 640
put() (*synapse.lib.cache.FixedCache method*), 641
put() (*synapse.lib.httpapi.ExtApiHandler method*), 704
put() (*synapse.lib.lmdbslab.MultiQueue method*), 731
put() (*synapse.lib.lmdbslab.Slab method*), 734
put() (*synapse.lib.queue.AQueue method*), 754
put() (*synapse.lib.queue.Queue method*), 754
put() (*synapse.lib.queue.Window method*), 755
put() (*synapse.lib.stormtypes.LibAxon method*), 792
putmulti() (*synapse.lib.lmdbslab.Slab method*), 734
puts() (*synapse.axon.Axon method*), 875
puts() (*synapse.axon.AxonApi method*), 881
puts() (*synapse.cryotank.CryoApi method*), 922
puts() (*synapse.cryotank.CryoTank method*), 924
puts() (*synapse.cryotank.TankApi method*), 925
puts() (*synapse.lib.lmdbslab.MultiQueue method*), 731
puts() (*synapse.lib.queue.Queue method*), 754
puts() (*synapse.lib.queue.Window method*), 755
putsQueue() (*synapse.lib.jsonstor.JsonStorApi method*), 712
putsQueue() (*synapse.lib.jsonstor.JsonStorCell method*), 713

Q

Query (*class in synapse.lib.ast*), 628
Query (*class in synapse.lib.stormtypes*), 803
query() (*synapse.lib.parser.Parser method*), 754
Queue (*class in synapse.lib.queue*), 754
Queue (*class in synapse.lib.stormtypes*), 804
queueLoop() (*synapse.cmds.cortex.Log method*), 570
QuitCmd (*class in synapse.tools.storm*), 865

R

raiseBadSyntax() (*synapse.lib.parser.AstConverter method*), 753
raisePermDeny() (*synapse.lib.hiveauth.HiveUser method*), 701
raises() (*synapse.tests.utils.SynTest method*), 850
Range (*class in synapse.lib.types*), 816
rangeexists() (*synapse.lib.lmdbslab.Slab method*), 734
RateLimit (*class in synapse.lib.ratelimit*), 755
RawPivot (*class in synapse.lib.ast*), 628
readlines() (*synapse.axon.Axon method*), 875
readlines() (*synapse.axon.AxonApi method*), 882
readlines() (*synapse.lib.stormtypes.LibAxon method*), 793
readonly (*synapse.lib.storm.Cmd attribute*), 772
readonly (*synapse.lib.storm.CountCmd attribute*), 773
readonly (*synapse.lib.storm.DiffCmd attribute*), 774
readonly (*synapse.lib.storm.IdenCmd attribute*), 776
readonly (*synapse.lib.storm.LimitCmd attribute*), 778
readonly (*synapse.lib.storm.MaxCmd attribute*), 778

readonly (*synapse.lib.storm.MinCmd attribute*), 779
readonly (*synapse.lib.storm.ParallelCmd attribute*), 781
readonly (*synapse.lib.storm.PureCmd attribute*), 782
readonly (*synapse.lib.storm.SleepCmd attribute*), 785
readonly (*synapse.lib.storm.SpinCmd attribute*), 785
readonly (*synapse.lib.storm.TeeCmd attribute*), 786
readonly (*synapse.lib.storm.TreeCmd attribute*), 787
readonly (*synapse.lib.storm.UniqCmd attribute*), 787
readonly (*synapse.lib.storm.ViewExecCmd attribute*), 788
readonly (*synapse.lib.stormlib.macro.MacroExecCmd attribute*), 597
readonly (*synapse.lib.stormlib.stats.StatsCountByCmd attribute*), 607
ReadOnlyLayer, 937
ReadOnlyProp, 937
readyToMirror() (*synapse.lib.cell.Cell method*), 654
readyToMirror() (*synapse.lib.cell.CellApi method*), 660
recover() (*synapse.lib.nexus.NexsRoot method*), 744
RecursionLimitHit, 937
recv() (*synapse.lib.link.Link method*), 728
recvsize() (*synapse.lib.link.Link method*), 728
redirectStdin() (*synapse.tests.utils.SynTest method*), 850
refang_text() (*in module synapse.lib.scrape*), 762
refang_text2() (*in module synapse.lib.scrape*), 762
regexizeEdgeGlob() (*in module synapse.lib.cache*), 642
regexizeTagGlob() (*in module synapse.lib.cache*), 642
registerLib() (*synapse.lib.stormtypes.StormTypesRegistry method*), 806
registerType() (*synapse.lib.stormtypes.StormTypesRegistry method*), 806
RegMethType (*class in synapse.lib.nexus*), 744
ReIndexCmd (*class in synapse.lib.storm*), 782
reload() (*synapse.lib.cell.Cell method*), 654
reload() (*synapse.lib.cell.CellApi method*), 660
ReloadCell (*class in synapse.tests.utils*), 840
RelProp (*class in synapse.lib.ast*), 628
RelPropCond (*class in synapse.lib.ast*), 628
RelPropValue (*class in synapse.lib.ast*), 629
rem() (*synapse.lib.cache.TagGlobs method*), 642
rem() (*synapse.lib.lmdbslab.MultiQueue method*), 731
rename() (*synapse.lib.hive.Hive method*), 695
renameVault() (*synapse.cortex.Cortex method*), 915
replace() (*synapse.lib.lmdbslab.Slab method*), 734
replace() (*synapse.lib.stormtypes.LibRegx method*), 797
replaceUnicodeDashes() (*in module synapse.lib.chop*), 678
replaceVaultConfigs() (*synapse.cortex.Cortex method*), 915
replaceVaultSecrets() (*synapse.cortex.Cortex method*), 916
reply() (*synapse.telepath.Task method*), 941
repr() (*in module synapse.lib.time*), 811
repr() (*synapse.lib.ast.AstNode method*), 618
repr() (*synapse.lib.ast.Const method*), 619
repr() (*synapse.lib.ast.List method*), 626
repr() (*synapse.lib.ast.N1Walk method*), 626
repr() (*synapse.lib.ast.PivotOper method*), 627
repr() (*synapse.lib.node.Node method*), 747
repr() (*synapse.lib.types.Array method*), 813
repr() (*synapse.lib.types.Bool method*), 813
repr() (*synapse.lib.types.Comp method*), 813
repr() (*synapse.lib.types.Duration method*), 814
repr() (*synapse.lib.types.Edge method*), 814
repr() (*synapse.lib.types.Float method*), 814
repr() (*synapse.lib.types.Int method*), 815
repr() (*synapse.lib.types.Ival method*), 816
repr() (*synapse.lib.types.Loc method*), 816
repr() (*synapse.lib.types.Ndef method*), 816
repr() (*synapse.lib.types.Range method*), 816
repr() (*synapse.lib.types.Str method*), 817
repr() (*synapse.lib.types.Taxonomy method*), 817
repr() (*synapse.lib.types.Time method*), 818
repr() (*synapse.lib.types.TimeEdge method*), 818
repr() (*synapse.lib.types.Type method*), 819
repr() (*synapse.models.geospace.Area method*), 830
repr() (*synapse.models.geospace.Dist method*), 830
repr() (*synapse.models.geospace.LatLong method*), 830
repr() (*synapse.models.inet.Fqdn method*), 831
repr() (*synapse.models.inet.IPv4 method*), 831
repr() (*synapse.models.infotech.SemVer method*), 834
repr() (*synapse.models.telco.Phone method*), 837
repr() (*synapse.tests.utils.TestSubType method*), 855
repr() (*synapse.tests.utils.ThreeType method*), 856
reprauthrule() (*in module synapse.common*), 893
reprNdef() (*in module synapse.lib.node*), 749
reprProp() (*in module synapse.lib.node*), 749
reprrule() (*in module synapse.tools.cellauth*), 861
reprs() (*synapse.lib.node.Node method*), 747
reprTag() (*in module synapse.lib.node*), 749
reprTagProps() (*in module synapse.lib.node*), 750
reqAdmin() (*synapse.lib.hiveauth.HiveUser method*), 701
reqAdmin() (*synapse.lib.storm.Runtime method*), 784
reqAhaProxy() (*synapse.lib.cell.Cell method*), 654
reqAuthAdmin() (*synapse.lib.httpapi.HandlerBase method*), 706
reqAuthGate() (*synapse.lib.hiveauth.Auth method*), 699
reqAuthUser() (*synapse.lib.httpapi.HandlerBase method*), 706
reqbytes() (*in module synapse.common*), 893

reqConfValid() (*synapse.lib.config.Config method*), 684
reqConfValu() (*synapse.lib.config.Config method*), 684
reqdir() (*in module synapse.common*), 893
reqfile() (*in module synapse.common*), 893
reqFormsByLook() (*synapse.datamodel.Model method*), 928
reqFormsByPrefix() (*synapse.datamodel.Model method*), 928
reqGateKeys() (*synapse.lib.cell.Cell method*), 654
reqGateKeys() (*synapse.lib.storm.Runtime method*), 784
reqjsonsafe() (*in module synapse.common*), 893
reqJsonSafeStrict() (*in module synapse.common*), 893
reqKeyValid() (*synapse.lib.config.Config method*), 684
reqNoParentQuorum() (*synapse.lib.view.View method*), 825
reqNotReadOnly() (*synapse.lib.nexus.NexsRoot method*), 744
reqParentQuorum() (*synapse.lib.view.View method*), 825
reqpath() (*in module synapse.common*), 893
reqPropsByLook() (*synapse.datamodel.Model method*), 928
reqRole() (*synapse.lib.hiveauth.Auth method*), 699
reqRoleByName() (*synapse.lib.hiveauth.Auth method*), 699
reqRuntSafe() (*synapse.lib.ast.AstNode method*), 618
reqStormMacro() (*synapse.cortex.Cortex method*), 916
reqUser() (*synapse.lib.hiveauth.Auth method*), 699
reqUserByName() (*synapse.lib.hiveauth.Auth method*), 699
reqUserByNameOrIden() (*synapse.lib.hiveauth.Auth method*), 699
reqUserCanReadLayer() (*synapse.lib.storm.Runtime method*), 784
reqValidStorm() (*synapse.cortex.CoreApi method*), 900
reqValidStorm() (*synapse.cortex.Cortex method*), 916
reqValidStormGraph() (*synapse.cortex.Cortex method*), 916
ReqValidStormV1 (*class in synapse.lib.httpapi*), 708
reqValidTdef() (*in module synapse.lib.trigger*), 812
reqValidVoter() (*synapse.lib.view.View method*), 825
reqVault() (*synapse.cortex.Cortex method*), 916
reqVaultByName() (*synapse.cortex.Cortex method*), 916
reqVaultByType() (*synapse.cortex.Cortex method*), 917
reqVersion() (*in module synapse.lib.version*), 822
reqView() (*synapse.cortex.Cortex method*), 917
result() (*in module synapse.common*), 893
result() (*synapse.telepath.Task method*), 941
resume() (*synapse.lib.lmdbslab.Scan method*), 732
resume() (*synapse.lib.lmdbslab.ScanBack method*), 732
resume() (*synapse.lib.lmdbslab.ScanKeys method*), 732
retnexc() (*in module synapse.common*), 893
Retry, 937
Return (*class in synapse.lib.ast*), 629
revCoreLayers() (*synapse.lib.modelrev.ModelRev method*), 737
reverseLift() (*synapse.lib.ast.LiftOper method*), 625
revModel20210126() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20210312() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20210528() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20210801() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20211112() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220307() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220315() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220509() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220706() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220803() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20220901() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20221025() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20221123() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20221212() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20221220() (*synapse.lib.modelrev.ModelRev method*), 737
revModel20230209() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_18() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_19() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_20() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_21() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_22() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_23() (*synapse.lib.modelrev.ModelRev method*), 737
revModel_0_2_24() (*synapse.lib.modelrev.ModelRev

method), 737
revoke() (*synapse.lib.certdir.Crl method*), 676
revoke() (*synapse.lib.hiveauth.HiveUser method*), 701
Rfc2822Addr (*class in synapse.models.inet*), 832
RiskModule (*class in synapse.models.risk*), 836
RobotHandler (*class in synapse.lib.httpapi*), 708
Role (*class in synapse.lib.stormlib.auth*), 583
role() (*synapse.lib.hiveauth.Auth method*), 699
roles() (*synapse.lib.hiveauth.Auth method*), 699
rotate() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 742
rotate() (*synapse.lib.nexus.NexsRoot method*), 744
rotateNexsLog() (*synapse.lib.cell.Cell method*), 654
rotateNexsLog() (*synapse.lib.cell.CellApi method*), 660
roundup() (*in module synapse.lib.stormlib.infosec*), 595
rows() (*synapse.cryotank.CryoApi method*), 922
rows() (*synapse.cryotank.CryoTank method*), 924
rows() (*synapse.lib.slabseqn.SlabSeqn method*), 765
RstHelp (*class in synapse.lib.autodoc*), 634
rtypes (*synapse.lib.stormtypes.StormTypesRegistry attribute*), 806
ruleFromText() (*in module synapse.lib.stormlib.auth*), 584
ruleFromText() (*synapse.lib.stormlib.auth.LibAuth static method*), 582
run() (*synapse.lib.ast.BreakOper method*), 619
run() (*synapse.lib.ast.CatchBlock method*), 619
run() (*synapse.lib.ast.CmdOper method*), 619
run() (*synapse.lib.ast.ContinueOper method*), 619
run() (*synapse.lib.ast.EditEdgeAdd method*), 620
run() (*synapse.lib.ast.EditEdgeDel method*), 620
run() (*synapse.lib.ast.EditNodeAdd method*), 620
run() (*synapse.lib.ast.EditParens method*), 620
run() (*synapse.lib.ast.EditPropDel method*), 620
run() (*synapse.lib.ast.EditPropSet method*), 620
run() (*synapse.lib.ast.EditTagAdd method*), 620
run() (*synapse.lib.ast.EditTagDel method*), 620
run() (*synapse.lib.ast.EditTagPropDel method*), 621
run() (*synapse.lib.ast.EditTagPropSet method*), 621
run() (*synapse.lib.ast.EditUnivDel method*), 621
run() (*synapse.lib.ast.Emit method*), 621
run() (*synapse.lib.ast.EmptyBlock method*), 621
run() (*synapse.lib.ast.FiltOper method*), 622
run() (*synapse.lib.ast.FiniBlock method*), 623
run() (*synapse.lib.ast.ForLoop method*), 623
run() (*synapse.lib.ast.FormPivot method*), 623
run() (*synapse.lib.ast.Function method*), 624
run() (*synapse.lib.ast.IfStmt method*), 624
run() (*synapse.lib.ast.InitBlock method*), 625
run() (*synapse.lib.ast.LiftOper method*), 625
run() (*synapse.lib.ast.Lookup method*), 626
run() (*synapse.lib.ast.N1Walk method*), 626
run() (*synapse.lib.ast.N1WalkNPivo method*), 626
run() (*synapse.lib.ast.N2WalkNPivo method*), 626
run() (*synapse.lib.ast.PivotIn method*), 627
run() (*synapse.lib.ast.PivotInFrom method*), 627
run() (*synapse.lib.ast.PivotOut method*), 627
run() (*synapse.lib.ast.PivotToTags method*), 627
run() (*synapse.lib.ast.PropPivot method*), 628
run() (*synapse.lib.ast.PropPivotOut method*), 628
run() (*synapse.lib.ast.Query method*), 628
run() (*synapse.lib.ast.RawPivot method*), 628
run() (*synapse.lib.ast.Return method*), 629
run() (*synapse.lib.ast.Search method*), 629
run() (*synapse.lib.ast.SetItemOper method*), 629
run() (*synapse.lib.ast.SetVarOper method*), 629
run() (*synapse.lib.ast.Stop method*), 629
run() (*synapse.lib.ast.SubGraph method*), 630
run() (*synapse.lib.ast.SubQuery method*), 630
run() (*synapse.lib.ast.SwitchCase method*), 631
run() (*synapse.lib.ast.TryCatch method*), 632
run() (*synapse.lib.ast.VarEvalOper method*), 632
run() (*synapse.lib.ast.VarListSetOper method*), 633
run() (*synapse.lib.ast.WhileLoop method*), 633
run() (*synapse.lib.ast.YieldValu method*), 633
run() (*synapse.lib.rstorm.StormRst method*), 757
run() (*synapse.lib.storm.StormDmon method*), 785
run_imap_coro() (*in module synapse.lib.stormlib.imap*), 594
runActiveTask() (*synapse.lib.cell.Cell method*), 654
RunAsCmd (*class in synapse.lib.storm*), 782
runBackup() (*synapse.lib.cell.Cell method*), 655
runBackup() (*synapse.lib.cell.CellApi method*), 660
runCmdLine() (*synapse.lib.cli.Cli method*), 680
runCmdLine() (*synapse.lib.cli.Cmd method*), 681
runCmdLine() (*synapse.lib.jupyter.CmdrCore method*), 714
runCmdLine() (*synapse.lib.jupyter.StormCore method*), 714
runCmdLine() (*synapse.lib.rstorm.StormOutput method*), 757
runCmdLine() (*synapse.tools.storm.StormCli method*), 866
runCmdLoop() (*synapse.lib.cli.Cli method*), 680
runCmdOpts() (*synapse.cmds.boss.KillCmd method*), 569
runCmdOpts() (*synapse.cmds.boss.PsCmd method*), 569
runCmdOpts() (*synapse.cmds.cortex.Log method*), 570
runCmdOpts() (*synapse.cmds.cortex.StormCmd method*), 571
runCmdOpts() (*synapse.cmds.hive.HiveCmd method*), 571
runCmdOpts() (*synapse.lib.cli.Cmd method*), 681
runCmdOpts() (*synapse.lib.cli.CmdHelp method*), 681
runCmdOpts() (*synapse.lib.cli.CmdLocals method*), 681
runCmdOpts() (*synapse.lib.cli.CmdQuit method*), 682

runCmdOpts() (*synapse.lib.rstorm.StormOutput method*), 757
runCmdOpts() (*synapse.tools.storm.ExportCmd method*), 864
runCmdOpts() (*synapse.tools.storm.PullFileCmd method*), 865
runCmdOpts() (*synapse.tools.storm.PushFileCmd method*), 865
runCmdOpts() (*synapse.tools.storm.RunFileCmd method*), 866
runcmdr() (*in module synapse.tools.cmdr*), 861
runCoreNodes() (*synapse.tests.utils.SynTest method*), 850
runCsvExport() (*in module synapse.tools.csvtool*), 861
runCsvImport() (*in module synapse.tools.csvtool*), 861
runDynTask() (*in module synapse.lib.dyndeps*), 690
runEdgeAdd() (*synapse.lib.trigger.Triggers method*), 812
runEdgeAdd() (*synapse.lib.view.View method*), 825
runEdgeDel() (*synapse.lib.trigger.Triggers method*), 812
runEdgeDel() (*synapse.lib.view.View method*), 825
RunFileCmd (*class in synapse.tools.storm*), 865
runGcCollect() (*synapse.lib.cell.CellApi method*), 660
runItemCmdr() (*in module synapse.lib.cmdr*), 682
runJsSchema() (*in module synapse.lib.stormlib.json*), 596
runLayrPull() (*synapse.cortex.Cortex method*), 917
runLayrPush() (*synapse.cortex.Cortex method*), 917
runloop() (*synapse.lib.agenda.Agenda method*), 612
runMirrorLoop() (*synapse.lib.nexus.NexsRoot method*), 744
runNodeAdd() (*synapse.lib.trigger.Triggers method*), 812
runNodeAdd() (*synapse.lib.view.View method*), 825
runNodeDel() (*synapse.lib.trigger.Triggers method*), 812
runNodeDel() (*synapse.lib.view.View method*), 825
runPropSet() (*synapse.lib.trigger.Triggers method*), 812
runPropSet() (*synapse.lib.view.View method*), 825
runRstCmdLine() (*synapse.lib.rstorm.StormCliOutput method*), 756
runRuntLift() (*synapse.cortex.Cortex method*), 917
runRuntPropDel() (*synapse.cortex.Cortex method*), 917
runRuntPropSet() (*synapse.cortex.Cortex method*), 917
runStorm() (*synapse.lib.modelrev.ModelRev method*), 737
runStormDmon() (*synapse.cortex.Cortex method*), 917
runStormSvcEvent() (*synapse.cortex.Cortex method*), 917
runTagAdd() (*synapse.lib.trigger.Triggers method*), 812
runTagAdd() (*synapse.lib.view.View method*), 825
runTagDel() (*synapse.lib.trigger.Triggers method*), 812
runTagDel() (*synapse.lib.view.View method*), 825
Runtime (*class in synapse.lib.storm*), 782
runtimeDocStormTypes() (*in module synapse.lib.autodoc*), 635
runtimeGetArgLines() (*in module synapse.lib.autodoc*), 635
runtimeGetReturnLines() (*in module synapse.lib.autodoc*), 635
runtopaque (*synapse.lib.ast.ArgvQuery attribute*), 618
runtopaque (*synapse.lib.ast.AstNode attribute*), 618
runtopaque (*synapse.lib.ast.EmbedQuery attribute*), 621
runtopaque (*synapse.lib.ast.Function attribute*), 624
runViewMerge() (*synapse.lib.view.View method*), 825
rx() (*synapse.lib.link.Link method*), 728
rx() (*synapse.lib.stormhttp.WebSocket method*), 790

S

sa_family (*synapse.lib.platforms.windows.sockaddr attribute*), 581
sanitizeUrl() (*in module synapse.lib.urlhelp*), 820
save() (*synapse.axon.Axon method*), 875
save() (*synapse.axon.UpLoad method*), 886
save() (*synapse.axon.UpLoadProxy method*), 886
save() (*synapse.cmds.cortex.Log method*), 570
save() (*synapse.lib.slabseqn.SlabSeqn method*), 765
save() (*synapse.lib.stormlib.yaml.LibYaml method*), 611
saveCaCert() (*synapse.lib.aha.AhaCell method*), 616
saveCaCertByts() (*synapse.lib.certdir.CertDir method*), 674
saveCertPem() (*synapse.lib.certdir.CertDir method*), 674
saveCodeCertBytes() (*synapse.lib.certdir.CertDir method*), 674
saveHiveTree() (*synapse.lib.cell.Cell method*), 655
saveHiveTree() (*synapse.lib.cell.CellApi method*), 660
saveHiveTree() (*synapse.lib.hive.Hive method*), 695
saveHiveTree() (*synapse.lib.hive.HiveApi method*), 695
saveHostCert() (*synapse.lib.aha.AhaCell method*), 616
saveHostCertByts() (*synapse.lib.certdir.CertDir method*), 674
saveLayerNodeEdits() (*synapse.cortex.CoreApi method*), 901
saveLayerNodeEdits() (*synapse.cortex.Cortex method*), 917
saveNodeEdits() (*synapse.lib.layer.Layer method*), 724
saveNodeEdits() (*synapse.lib.layer.LayerApi method*), 725
saveNodeEdits() (*synapse.lib.snap.Snap method*), 769

saveNodeEdits() (*synapse.lib.view.ViewApi method*), 826
savePkeyPem() (*synapse.lib.certdir.CertDir method*), 674
saveto() (*synapse.lib.lmdbslab.LmdbBackup method*), 731
saveToNexs() (*synapse.lib.nexus.Pusher method*), 744
saveUserCert() (*synapse.lib.aha.AhaCell method*), 616
saveUserCertByts() (*synapse.lib.certdir.CertDir method*), 674
saveVectToNode() (*synapse.lib.stormlib.infosec.CvssLib method*), 594
Scan (*class in synapse.lib.lmdbslab*), 731
ScanBack (*class in synapse.lib.lmdbslab*), 732
scanByDups() (*synapse.lib.layer.IndxBy method*), 719
scanByDups() (*synapse.lib.lmdbslab.Slab method*), 734
scanByDupsBack() (*synapse.lib.lmdbslab.Slab method*), 734
scanByFull() (*synapse.lib.lmdbslab.Slab method*), 734
scanByFullBack() (*synapse.lib.lmdbslab.Slab method*), 734
scanByPref() (*synapse.lib.layer.IndxBy method*), 719
scanByPref() (*synapse.lib.lmdbslab.Slab method*), 734
scanByPrefBack() (*synapse.lib.layer.IndxBy method*), 719
scanByPrefBack() (*synapse.lib.lmdbslab.Slab method*), 735
scanByRange() (*synapse.lib.layer.IndxBy method*), 719
scanByRange() (*synapse.lib.lmdbslab.Slab method*), 735
scanByRangeBack() (*synapse.lib.layer.IndxBy method*), 719
scanByRangeBack() (*synapse.lib.lmdbslab.Slab method*), 735
ScanKeys (*class in synapse.lib.lmdbslab*), 732
scanKeys() (*synapse.lib.lmdbslab.Slab method*), 735
scanKeysByPref() (*synapse.lib.lmdbslab.Slab method*), 735
schedCallSafe() (*synapse.lib.base.Base method*), 637
schedCoro() (*synapse.lib.base.Base method*), 638
schedCoroSafe() (*synapse.lib.base.Base method*), 638
schedCoroSafePend() (*synapse.lib.base.Base method*), 638
schedGenr() (*in module synapse.lib.base*), 640
SchemaViolation, 937
ScienceModule (*class in synapse.models.science*), 836
scol (*synapse.lib.parser.AstInfo attribute*), 753
Scope (*class in synapse.lib.scope*), 757
scrape() (*in module synapse.lib.scrape*), 763
scrapeAsync() (*in module synapse.lib.scrape*), 763
ScrapeCmd (*class in synapse.lib.storm*), 784
scrapeIface() (*synapse.lib.view.View method*), 825
scrub() (*synapse.lib.snap.Scrubber method*), 767
Scrubber (*class in synapse.lib.snap*), 766
scrubLines() (*in module synapse.lib.autodoc*), 635
Search (*class in synapse.lib.ast*), 629
search() (*synapse.lib.parser.Parser method*), 754
search() (*synapse.lib.stormlib.imap.ImapServer method*), 594
search() (*synapse.lib.stormtypes.LibRegx method*), 797
second() (*in module synapse.lib.time*), 811
second() (*synapse.lib.stormtypes.LibTime method*), 799
select() (*synapse.lib.stormlib.imap.ImapServer method*), 594
selfSignCert() (*synapse.lib.certdir.CertDir method*), 674
semafork() (*in module synapse.lib.coro*), 688
SemVer (*class in synapse.models.infotech*), 834
send() (*synapse.lib.link.Link method*), 728
send() (*synapse.lib.stormlib.smtp.SmtpMessage method*), 605
sendAuthRequired() (*synapse.lib.httpapi.HandlerBase method*), 706
sendRestErr() (*synapse.lib.httpapi.HandlerBase method*), 706
sendRestExc() (*synapse.lib.httpapi.HandlerBase method*), 707
sendRestRetn() (*synapse.lib.httpapi.HandlerBase method*), 707
serialize() (*in module synapse.tools.healthcheck*), 863
Service (*class in synapse.lib.stormtypes*), 804
Sess (*class in synapse.daemon*), 925
Sess (*class in synapse.lib.httpapi*), 708
sess() (*synapse.lib.httpapi.HandlerBase method*), 707
Set (*class in synapse.lib.spooled*), 770
Set (*class in synapse.lib.stormtypes*), 804
set() (*in module synapse.lib.scope*), 759
set() (*synapse.exc.SynErr method*), 938
set() (*synapse.lib.cli.Cli method*), 680
set() (*synapse.lib.hive.Hive method*), 695
set() (*synapse.lib.hive.HiveDict method*), 696
set() (*synapse.lib.hive.Node method*), 696
set() (*synapse.lib.hive.TeleHive method*), 697
set() (*synapse.lib.httpapi.Sess method*), 708
set() (*synapse.lib.link.Link method*), 728
set() (*synapse.lib.lmdbslab.GuidStor method*), 729
set() (*synapse.lib.lmdbslab.HotCount method*), 730
set() (*synapse.lib.lmdbslab.HotKeyVal method*), 730
set() (*synapse.lib.lmdbslab.SlabDict method*), 736
set() (*synapse.lib.node.Node method*), 747
set() (*synapse.lib.scope.Scope method*), 758
set() (*synapse.lib.slaboffs.SlabOffs method*), 764
set() (*synapse.lib.snap.ProtoNode method*), 766
set() (*synapse.lib.spooled.Dict method*), 770
set() (*synapse.lib.stormlib.auth.UserJson method*), 584
set() (*synapse.lib.stormtypes.LibJsonStor method*), 795
set() (*synapse.lib.stormtypes.NodeProps method*), 801

set() (*synapse.lib.stormtypes.Trigger method*), 807
set() (*synapse.lib.trigger.Trigger method*), 812
set_default_headers() (*synapse.lib.httpapi.ExtApiHandler method*), 704
set_default_headers() (*synapse.lib.httpapi.HandlerBase method*), 707
set_inputs() (*synapse.lib.storm.Parser method*), 781
set_key() (*synapse.lib.lmdbslab.Scan method*), 732
set_key() (*synapse.lib.lmdbslab.ScanBack method*), 732
set_pool_logging() (*in module synapse.lib.coro*), 688
set_range() (*synapse.lib.lmdbslab.Scan method*), 732
set_range() (*synapse.lib.lmdbslab.ScanBack method*), 732
setAdmin() (*synapse.lib.hiveauth.HiveUser method*), 701
setAhaSvcDown() (*synapse.lib.aha.AhaCell method*), 616
setAndSync() (*synapse.lib.hive.HiveApi method*), 695
setArchived() (*synapse.lib.hiveauth.HiveUser method*), 701
setArgv() (*synapse.lib.storm.Cmd method*), 772
setAuthAdmin() (*synapse.lib.cell.CellApi method*), 660
setBytsToAbrv() (*synapse.lib.lmdbslab.SlabAbrv method*), 735
setCellActive() (*synapse.lib.cell.Cell method*), 655
setCellUser() (*synapse.lib.cell.CellApi method*), 660
setCmprCtor() (*synapse.lib.types.Type method*), 819
setConfFromEnvs() (*synapse.lib.config.Config method*), 685
setConfFromFile() (*synapse.lib.config.Config method*), 685
setConfFromOpts() (*synapse.lib.config.Config method*), 685
setData() (*synapse.lib.node.Node method*), 747
setData() (*synapse.lib.snap.ProtoNode method*), 766
setdefault() (*synapse.exc.SynErr method*), 938
setdefault() (*synapse.lib.hive.HiveDict method*), 696
setDeprLock() (*synapse.cortex.Cortex method*), 917
setFeedFunc() (*synapse.cortex.Cortex method*), 917
setGraph() (*synapse.lib.storm.Runtime method*), 784
setGreedCoro() (*in module synapse.glob*), 938
setHiveKey() (*synapse.lib.cell.Cell method*), 655
setHiveKey() (*synapse.lib.cell.CellApi method*), 661
setHttpApiIndx() (*synapse.cortex.Cortex method*), 918
setHttpApiIndx() (*synapse.lib.stormlib.cortex.CortexHttpApi method*), 587
setHttpSessInfo() (*synapse.lib.cell.Cell method*), 655
setIndex() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 743
setindex() (*synapse.lib.nexus.NexsRoot method*), 744
setitem() (*synapse.lib.stormlib.auth.UserProfile method*), 584
setitem() (*synapse.lib.stormlib.auth.UserVars method*), 584
setitem() (*synapse.lib.stormlib.cortex.HttpApiVars method*), 589
setitem() (*synapse.lib.stormlib.cortex.HttpHeaderDict method*), 590
setitem() (*synapse.lib.stormlib.cortex.HttpPermsList method*), 590
setitem() (*synapse.lib.stormlib.vault.VaultConfigs method*), 609
setitem() (*synapse.lib.stormlib.vault.VaultSecrets method*), 609
setitem() (*synapse.lib.stormtypes.CmdOpts method*), 790
setitem() (*synapse.lib.stormtypes.Dict method*), 791
setitem() (*synapse.lib.stormtypes.List method*), 800
setitem() (*synapse.lib.stormtypes.NodeProps method*), 801
setitem() (*synapse.lib.stormtypes.PathMeta method*), 802
setitem() (*synapse.lib.stormtypes.PathVars method*), 802
setitem() (*synapse.lib.stormtypes.StormType method*), 805
SetItemOper (*class in synapse.lib.ast*), 629
setJsonObj() (*synapse.cortex.Cortex method*), 918
setJsonObjProp() (*synapse.cortex.Cortex method*), 918
setLayerInfo() (*synapse.lib.layer.Layer method*), 724
setLayers() (*synapse.lib.view.View method*), 825
setLiftHintCmprCtor() (*synapse.lib.types.Type method*), 820
setLocked() (*synapse.lib.hiveauth.HiveUser method*), 701
setlogging() (*in module synapse.common*), 893
setMergeComment() (*synapse.lib.stormtypes.View method*), 807
setMergeComment() (*synapse.lib.view.View method*), 825
setMergeRequest() (*synapse.lib.stormtypes.View method*), 807
setMergeRequest() (*synapse.lib.view.View method*), 825
setMergeVote() (*synapse.lib.stormtypes.View method*), 807
setMergeVote() (*synapse.lib.view.View method*), 825
setMergeVoteComment() (*synapse.lib.stormtypes.View method*), 807
setMergeVoteComment() (*synapse.lib.view.View method*), 825
setMesg() (*synapse.tests.utils.AsyncStreamEvent method*), 839

setMesg() (*synapse.tests.utils.StreamEvent method*), 841
setModelVers() (*synapse.lib.layer.Layer method*), 724
setName() (*synapse.lib.hiveauth.HiveRole method*), 700
setName() (*synapse.lib.hiveauth.HiveUser method*), 701
setNexsIndx() (*synapse.lib.cell.Cell method*), 655
setNexsReady() (*synapse.lib.nexus.NexsRoot method*), 744
setNexsRoot() (*synapse.lib.nexus.Pusher method*), 744
setNormFunc() (*synapse.lib.types.Type method*), 820
setOAuthAuthCode() (*synapse.lib.oauth.OAuthMixin method*), 752
setOpt() (*synapse.lib.storm.Runtime method*), 784
setPasswd() (*synapse.lib.hiveauth.HiveUser method*), 701
setPathLink() (*synapse.lib.jsonstor.JsonStor method*), 711
setPathLink() (*synapse.lib.jsonstor.JsonStorApi method*), 712
setPathLink() (*synapse.lib.jsonstor.JsonStorCell method*), 713
setPathObj() (*synapse.lib.jsonstor.JsonStor method*), 711
setPathObj() (*synapse.lib.jsonstor.JsonStorApi method*), 712
setPathObj() (*synapse.lib.jsonstor.JsonStorCell method*), 713
setPathObjProp() (*synapse.lib.jsonstor.JsonStor method*), 711
setPathObjProp() (*synapse.lib.jsonstor.JsonStorApi method*), 712
setPathObjProp() (*synapse.lib.jsonstor.JsonStorCell method*), 713
setProcName() (*in module synapse.lib.platforms.common*), 579
setProp() (*synapse.datamodel.Form method*), 926
setPropAbrv() (*synapse.lib.layer.Layer method*), 724
setReady() (*synapse.daemon.Daemon method*), 925
setRoleInfo() (*synapse.lib.hiveauth.Auth method*), 699
setRoleName() (*synapse.lib.cell.Cell method*), 655
setRoleName() (*synapse.lib.hiveauth.Auth method*), 699
setRoleRules() (*synapse.lib.cell.Cell method*), 655
setRoleRules() (*synapse.lib.cell.CellApi method*), 661
setRoles() (*synapse.lib.hiveauth.HiveUser method*), 701
setRules() (*synapse.lib.hiveauth.HiveRuler method*), 700
sets() (*synapse.lib.lmdbslab.MultiQueue method*), 731
setSessItem() (*synapse.daemon.Sess method*), 925
setSodeDirty() (*synapse.lib.layer.Layer method*), 724
setStatus() (*synapse.lib.health.HealthCheck method*), 693
setStormCmd() (*synapse.cortex.Cortex method*), 918
setStormGraphPerm() (*synapse.cortex.Cortex method*), 918
setStormMacroPerm() (*synapse.cortex.Cortex method*), 918
setStormPool() (*synapse.cortex.Cortex method*), 918
setStormSvcEvents() (*synapse.cortex.Cortex method*), 918
setStormVar() (*synapse.cortex.CoreApi method*), 901
setStormVar() (*synapse.cortex.Cortex method*), 918
setSynDir() (*synapse.tests.utils.SynTest method*), 850
setTagModel() (*synapse.cortex.Cortex method*), 918
setTagProp() (*synapse.lib.node.Node method*), 747
setTagProp() (*synapse.lib.snap.ProtoNode method*), 766
setTagPropAbrv() (*synapse.lib.layer.Layer method*), 724
setTriggerInfo() (*synapse.lib.view.View method*), 825
setTstEnvars() (*synapse.tests.utils.SynTest method*), 850
setup() (*in module synapse.tools.pullfile*), 863
setUserAdmin() (*synapse.lib.cell.Cell method*), 655
setUserAdmin() (*synapse.lib.cell.CellApi method*), 661
setUserArchived() (*synapse.lib.cell.Cell method*), 655
setUserArchived() (*synapse.lib.cell.CellApi method*), 661
setUserEmail() (*synapse.lib.cell.Cell method*), 655
setUserEmail() (*synapse.lib.cell.CellApi method*), 661
setUserInfo() (*synapse.lib.hiveauth.Auth method*), 700
setUserLocked() (*synapse.cortex.Cortex method*), 919
setUserLocked() (*synapse.lib.cell.Cell method*), 655
setUserLocked() (*synapse.lib.cell.CellApi method*), 661
setUserName() (*synapse.lib.cell.Cell method*), 655
setUserName() (*synapse.lib.hiveauth.Auth method*), 700
setUserPasswd() (*synapse.lib.cell.Cell method*), 655
setUserPasswd() (*synapse.lib.cell.CellApi method*), 661
setUserProfInfo() (*synapse.lib.cell.Cell method*), 655
setUserProfInfo() (*synapse.lib.cell.CellApi method*), 661
setUserRoles() (*synapse.lib.cell.Cell method*), 655
setUserRoles() (*synapse.lib.cell.CellApi method*), 661
setUserRules() (*synapse.lib.cell.Cell method*), 655
setUserRules() (*synapse.lib.cell.CellApi method*), 661
setUserVarValu() (*synapse.lib.cell.Cell method*), 655
setVar() (*synapse.lib.node.Path method*), 748
setVar() (*synapse.lib.storm.Runtime method*), 784
SetVarOper (*class in synapse.lib.ast*), 629
setVaultConfigs() (*synapse.cortex.Cortex method*), 919

setVaultPerm() (*synapse.cortex.Cortex method*), 919
setVaultSecrets() (*synapse.cortex.Cortex method*), 919
setViewInfo() (*synapse.lib.view.View method*), 825
setViewLayers() (*synapse.cortex.Cortex method*), 920
Share (*class in synapse.lib.share*), 764
Share (*class in synapse.telepath*), 941
share() (*synapse.daemon.Daemon method*), 925
sibling() (*synapse.lib.ast.AstNode method*), 618
sign() (*synapse.lib.crypto.ecc.PriKey method*), 573
sign() (*synapse.lib.crypto.rsa.PriKey method*), 576
signCertAs() (*synapse.lib.certdir.CertDir method*), 675
signedint64en() (*in module synapse.common*), 894
signedint64un() (*in module synapse.common*), 894
signext() (*synapse.lib.stormlib.hex.HexLib method*), 593
signHostCsr() (*synapse.lib.aha.AhaApi method*), 614
signHostCsr() (*synapse.lib.aha.AhaCell method*), 616
signHostCsr() (*synapse.lib.aha.ProvApi method*), 617
signHostCsr() (*synapse.lib.certdir.CertDir method*), 675
signitem() (*synapse.lib.crypto.rsa.PriKey method*), 576
signUserCsr() (*synapse.lib.aha.AhaApi method*), 614
signUserCsr() (*synapse.lib.aha.AhaCell method*), 616
signUserCsr() (*synapse.lib.aha.EnrollApi method*), 617
signUserCsr() (*synapse.lib.aha.ProvApi method*), 617
signUserCsr() (*synapse.lib.certdir.CertDir method*), 675
size() (*synapse.axon.Axon method*), 876
size() (*synapse.axon.AxonApi method*), 882
size() (*synapse.lib.lmdbslab.MultiQueue method*), 731
size() (*synapse.lib.queue.Queue method*), 754
size() (*synapse.lib.stormlib.stix.StixBundle method*), 608
size() (*synapse.lib.stormtypes.LibAxon method*), 793
size() (*synapse.telepath.ClientV2 method*), 940
skip() (*synapse.tests.utils.SynTest method*), 851
skipIfNexusReplay() (*synapse.tests.utils.SynTest method*), 851
skipIfNoInternet() (*synapse.tests.utils.SynTest method*), 851
skipIfNoPath() (*synapse.tests.utils.SynTest method*), 851
skipLongTest() (*synapse.tests.utils.SynTest method*), 851
Slab (*class in synapse.lib.lmdbslab*), 732
SlabAbrv (*class in synapse.lib.lmdbslab*), 735
SlabAlreadyOpen, 937
SlabDict (*class in synapse.lib.lmdbslab*), 735
slabFilename() (*synapse.lib.multislabseqn.MultiSlabSeqn static method*), 743
SlabHive (*class in synapse.lib.hive*), 696
SlabInUse, 937
SlabOffs (*class in synapse.lib.slaboffs*), 764
SlabSeqn (*class in synapse.lib.slabseqn*), 764
SleepCmd (*class in synapse.lib.storm*), 785
slice() (*synapse.cryotank.CryoApi method*), 922
slice() (*synapse.cryotank.CryoTank method*), 924
slice() (*synapse.cryotank.TankApi method*), 925
slice() (*synapse.lib.queue.AQueue method*), 754
slice() (*synapse.lib.queue.Queue method*), 754
slice() (*synapse.lib.slabseqn.SlabSeqn method*), 766
sliceBack() (*synapse.lib.slabseqn.SlabSeqn method*), 766
slices() (*synapse.lib.queue.Queue method*), 755
sline (*synapse.lib.parser.AstInfo attribute*), 753
SmtpLib (*class in synapse.lib.stormlib.smtp*), 605
SmtpMessage (*class in synapse.lib.stormlib.smtp*), 605
Snap (*class in synapse.lib.snap*), 767
snap() (*synapse.cortex.Cortex method*), 920
snap() (*synapse.lib.view.View method*), 826
snapctor() (*synapse.lib.view.View class method*), 826
SnapEditor (*class in synapse.lib.snap*), 770
sockaddr (*class in synapse.lib.platforms.windows*), 581
soff (*synapse.lib.parser.AstInfo attribute*), 753
someargs() (*synapse.tests.utils.LibTst method*), 840
sorted() (*synapse.lib.stormlib.stats.StatTally method*), 606
sorteq() (*synapse.tests.utils.SynTest method*), 851
source() (*synapse.utils.stormcov.plugin.StormReporter method*), 870
spawn() (*in module synapse.lib.coro*), 688
SpawnExit, 937
spin() (*in module synapse.common*), 894
spin() (*synapse.lib.coro.GenrHelp method*), 687
SpinCmd (*class in synapse.lib.storm*), 785
Spooled (*class in synapse.lib.spooled*), 770
SpooledSet (*class in synapse.lib.stormlib.spooled*), 605
SSLCTX_CACHE_SIZE (*in module synapse.lib.cell*), 662
stablebuid() (*synapse.tests.utils.SynTest method*), 851
stableguid() (*synapse.tests.utils.SynTest method*), 851
start() (*synapse.lib.storm.DmonManager method*), 775
startup() (*synapse.lib.nexus.NexsRoot method*), 744
stat() (*synapse.lib.layer.Layer method*), 724
stat() (*synapse.lib.lmdbslab.Slab method*), 735
stat() (*synapse.lib.slabseqn.SlabSeqn method*), 766
statinfo() (*synapse.lib.lmdbslab.Slab method*), 735
StatsCountByCmd (*class in synapse.lib.stormlib.stats*), 606
StatTally (*class in synapse.lib.stormlib.stats*), 606
status() (*synapse.lib.lmdbslab.MultiQueue method*), 731
stems() (*synapse.lib.types.Loc method*), 816
StepTimeout, 937
StixBundle (*class in synapse.lib.stormlib.stix*), 608

Stop (*class in synapse.lib.ast*), 629
stop() (*synapse.lib.storm.DmonManager method*), 775
stop() (*synapse.lib.storm.StormDmon method*), 786
storm() (*synapse.cortex.CoreApi method*), 901
storm() (*synapse.cortex.Cortex method*), 920
storm() (*synapse.lib.jupyter.CmdrCore method*), 714
storm() (*synapse.lib.jupyter.StormCore method*), 714
storm() (*synapse.lib.node.Node method*), 747
storm() (*synapse.lib.snap.Snap method*), 769
storm() (*synapse.lib.storm.Runtime method*), 784
storm() (*synapse.lib.view.View method*), 826
storm() (*synapse.tools.storm.StormCli method*), 866
storm_prefix (*synapse.lib.httpapi.ExtApiHandler attribute*), 704
StormBreak, 788
StormCallV1 (*class in synapse.lib.httpapi*), 708
StormCli (*class in synapse.tools.storm*), 866
StormCliCmd (*class in synapse.tools.storm*), 866
StormCliOutput (*class in synapse.lib.rstorm*), 756
StormCmd (*class in synapse.cmds.cortex*), 570
stormcmdargs() (*synapse.lib.parser.AstConverter method*), 753
StormCompleter (*class in synapse.tools.storm*), 867
StormContinue, 788
StormCore (*class in synapse.lib.jupyter*), 714
StormCtrlFlow, 788
StormCtrlTracer (*class in synapse.utils.stormcov.plugin*), 868
StormDmon (*class in synapse.lib.storm*), 785
StormExit, 788
StormExportV1 (*class in synapse.lib.httpapi*), 708
stormfunc() (*in module synapse.lib.stormtypes*), 808
StormHandler (*class in synapse.lib.httpapi*), 709
stormHasNoErr() (*synapse.tests.utils.SynTest method*), 851
stormHasNoWarnErr() (*synapse.tests.utils.SynTest method*), 852
StormHiveDict (*class in synapse.lib.stormtypes*), 804
stormIsInErr() (*synapse.tests.utils.SynTest method*), 852
stormIsInPrint() (*synapse.tests.utils.SynTest method*), 852
stormIsInWarn() (*synapse.tests.utils.SynTest method*), 852
StormLexer (*class in synapse.lib.storm_format*), 788
stormlist() (*synapse.cortex.Cortex method*), 920
stormlist() (*synapse.lib.view.View method*), 826
stormlogger (*in module synapse.cortex*), 922
StormNodesV1 (*class in synapse.lib.httpapi*), 709
stormNotInPrint() (*synapse.tests.utils.SynTest method*), 852
stormNotInWarn() (*synapse.tests.utils.SynTest method*), 852
StormOutput (*class in synapse.lib.rstorm*), 756
StormPkgConflicts, 937
StormPkgRequires, 937
StormPkgTest (*class in synapse.tests.utils*), 840
StormPlugin (*class in synapse.utils.stormcov.plugin*), 869
StormPoolDelCmd (*class in synapse.lib.stormlib.cortex*), 590
StormPoolGetCmd (*class in synapse.lib.stormlib.cortex*), 590
StormPoolSetCmd (*class in synapse.lib.stormlib.cortex*), 590
StormRaise, 937
StormReporter (*class in synapse.utils.stormcov.plugin*), 870
stormrepr() (*synapse.lib.stormlib.aha.AhaPool method*), 581
stormrepr() (*synapse.lib.stormlib.auth.Role method*), 583
stormrepr() (*synapse.lib.stormlib.auth.User method*), 583
stormrepr() (*synapse.lib.stormlib.cache.FixedCache method*), 585
stormrepr() (*synapse.lib.stormlib.cortex.HttpApi method*), 588
stormrepr() (*synapse.lib.stormlib.json.JsonSchema method*), 596
stormrepr() (*synapse.lib.stormlib.spooled.SpooledSet method*), 605
stormrepr() (*synapse.lib.stormlib.vault.Vault method*), 609
stormrepr() (*synapse.lib.stormlib.vault.VaultConfigs method*), 609
stormrepr() (*synapse.lib.stormtypes.CmdOpts method*), 790
stormrepr() (*synapse.lib.stormtypes.Dict method*), 791
stormrepr() (*synapse.lib.stormtypes.Lib method*), 792
stormrepr() (*synapse.lib.stormtypes.List method*), 800
stormrepr() (*synapse.lib.stormtypes.Number method*), 801
stormrepr() (*synapse.lib.stormtypes.Prim method*), 802
stormrepr() (*synapse.lib.stormtypes.Proxy method*), 803
stormrepr() (*synapse.lib.stormtypes.ProxyGenrMethod method*), 803
stormrepr() (*synapse.lib.stormtypes.ProxyMethod method*), 803
stormrepr() (*synapse.lib.stormtypes.Query method*), 804
stormrepr() (*synapse.lib.stormtypes.Queue method*), 804
stormrepr() (*synapse.lib.stormtypes.Set method*), 804
stormrepr() (*synapse.lib.stormtypes.Undef method*), 807
StormReturn, 788

StormRst (*class in synapse.lib.rstorm*), 757
StormRuntimeError, 937
StormStop, 788
stormstring() (*in module synapse.lib.chop*), 678
StormSvc (*class in synapse.lib.stormsvc*), 790
StormSvcClient (*class in synapse.lib.stormsvc*), 790
StormType (*class in synapse.lib.stormtypes*), 805
StormTypesRegistry (*class in synapse.lib.stormtypes*), 805
StormV1 (*class in synapse.lib.httpapi*), 709
StormVarListError, 937
StormVarsGetV1 (*class in synapse.lib.httpapi*), 709
StormVarsPopV1 (*class in synapse.lib.httpapi*), 709
StormVarsSetV1 (*class in synapse.lib.httpapi*), 709
storNodeDele() (*synapse.lib.hive.Hive method*), 695
storNodeDele() (*synapse.lib.hive.SlabHive method*), 696
storNodeEdits() (*synapse.lib.layer.Layer method*), 724
storNodeEdits() (*synapse.lib.layer.LayerApi method*), 725
storNodeEdits() (*synapse.lib.view.View method*), 826
storNodeEdits() (*synapse.lib.view.ViewApi method*), 826
storNodeEditsNoLift() (*synapse.lib.layer.Layer method*), 724
storNodeEditsNoLift() (*synapse.lib.layer.LayerApi method*), 726
storNodeValu() (*synapse.lib.hive.Hive method*), 695
storNodeValu() (*synapse.lib.hive.SlabHive method*), 697
StorType (*class in synapse.lib.layer*), 726
stortype (*synapse.lib.types.Bool attribute*), 813
stortype (*synapse.lib.types.Comp attribute*), 813
stortype (*synapse.lib.types.Data attribute*), 814
stortype (*synapse.lib.types.Duration attribute*), 814
stortype (*synapse.lib.types.Edge attribute*), 814
stortype (*synapse.lib.types.Float attribute*), 814
stortype (*synapse.lib.types.Guid attribute*), 814
stortype (*synapse.lib.types.Hex attribute*), 815
stortype (*synapse.lib.types.HugeNum attribute*), 815
stortype (*synapse.lib.types.Ival attribute*), 816
stortype (*synapse.lib.types.Loc attribute*), 816
stortype (*synapse.lib.types.Ndef attribute*), 816
stortype (*synapse.lib.types.NodeProp attribute*), 816
stortype (*synapse.lib.types.Range attribute*), 817
stortype (*synapse.lib.types.Str attribute*), 817
stortype (*synapse.lib.types.Time attribute*), 818
stortype (*synapse.lib.types.TimeEdge attribute*), 818
stortype (*synapse.lib.types.Type attribute*), 820
stortype (*synapse.lib.types.Velocity attribute*), 820
stortype (*synapse.models.geospace.LatLong attribute*), 831
stortype (*synapse.models.inet.Fqdn attribute*), 831
stortype (*synapse.models.inet.IPv4 attribute*), 832
stortype (*synapse.models.inet.IPv6 attribute*), 832
stortype (*synapse.tests.utils.TestSubType attribute*), 855
stortype (*synapse.tests.utils.TestType attribute*), 855
stortype (*synapse.tests.utils.ThreeType attribute*), 856
StorTypeFloat (*class in synapse.lib.layer*), 726
StorTypeFqdn (*class in synapse.lib.layer*), 726
StorTypeGuid (*class in synapse.lib.layer*), 726
StorTypeHier (*class in synapse.lib.layer*), 727
StorTypeHugeNum (*class in synapse.lib.layer*), 727
StorTypeInt (*class in synapse.lib.layer*), 727
StorTypeIpv6 (*class in synapse.lib.layer*), 727
StorTypeIval (*class in synapse.lib.layer*), 727
StorTypeLatLon (*class in synapse.lib.layer*), 727
StorTypeLoc (*class in synapse.lib.layer*), 727
StorTypeMsgp (*class in synapse.lib.layer*), 727
StorTypeTag (*class in synapse.lib.layer*), 728
StorTypeTime (*class in synapse.lib.layer*), 728
StorTypeUtf8 (*class in synapse.lib.layer*), 728
Str (*class in synapse.lib.stormtypes*), 806
Str (*class in synapse.lib.types*), 817
StreamEvent (*class in synapse.tests.utils*), 841
StreamHandler (*class in synapse.lib.httpapi*), 709
strify() (*synapse.lib.stormhttp.LibHttp method*), 789
strify() (*synapse.lib.stormtypes.LibAxon method*), 793
SubGraph (*class in synapse.lib.ast*), 629
SubqCond (*class in synapse.lib.ast*), 630
SubQuery (*class in synapse.lib.ast*), 630
subquery() (*synapse.lib.parser.AstConverter method*), 753
substrate_check() (*in module synapse.lib.crypto.coin*), 572
suppress_logging() (*in module synapse.lib.jupyter*), 718
suppress_logging() (*synapse.lib.jupyter.CmdrCore method*), 714
suppress_logging() (*synapse.lib.jupyter.StormCore method*), 715
svciden (*synapse.lib.storm.Cmd attribute*), 772
SwitchCase (*class in synapse.lib.ast*), 630
switchcase() (*synapse.lib.parser.AstConverter method*), 753
switchext() (*in module synapse.common*), 894
synapse
    module, 569
synapse.axon
    module, 871
synapse.cells
    module, 886
synapse.cmds
    module, 569
synapse.cmds.boss
    module, 569
synapse.cmds.cortex

module, 570
synapse.cmds.hive
module, 571
synapse.common
module, 886
synapse.cortex
module, 895
synapse.cryotank
module, 922
synapse.daemon
module, 925
synapse.data
module, 571
synapse.datamodel
module, 926
synapse.exc
module, 930
synapse.glob
module, 938
synapse.lib
module, 572
synapse.lib.agenda
module, 611
synapse.lib.aha
module, 613
synapse.lib.ast
module, 617
synapse.lib.autodoc
module, 634
synapse.lib.base
module, 635
synapse.lib.boss
module, 641
synapse.lib.cache
module, 641
synapse.lib.cell
module, 642
synapse.lib.certdir
module, 662
synapse.lib.chop
module, 677
synapse.lib.cli
module, 679
synapse.lib.cmd
module, 682
synapse.lib.cmdr
module, 682
synapse.lib.config
module, 683
synapse.lib.const
module, 687
synapse.lib.coro
module, 687
synapse.lib.crypto
module, 572
synapse.lib.crypto.coin
module, 572
synapse.lib.crypto.ecc
module, 572
synapse.lib.crypto.passwd
module, 575
synapse.lib.crypto.rsa
module, 576
synapse.lib.crypto.tinfoil
module, 578
synapse.lib.datfile
module, 689
synapse.lib.dyndeps
module, 689
synapse.lib.encoding
module, 690
synapse.lib.gis
module, 691
synapse.lib.grammar
module, 692
synapse.lib.hashitem
module, 693
synapse.lib.hashset
module, 693
synapse.lib.health
module, 693
synapse.lib.hive
module, 694
synapse.lib.hiveauth
module, 697
synapse.lib.httpapi
module, 702
synapse.lib.ingest
module, 710
synapse.lib.interval
module, 710
synapse.lib.jsonstor
module, 711
synapse.lib.jupyter
module, 713
synapse.lib.layer
module, 718
synapse.lib.link
module, 728
synapse.lib.lmdbslab
module, 729
synapse.lib.modelrev
module, 737
synapse.lib.module
module, 738
synapse.lib.modules
module, 740
synapse.lib.msgpack

module, 740
synapse.lib.multislabseqn
module, 742
synapse.lib.nexus
module, 743
synapse.lib.node
module, 745
synapse.lib.oauth
module, 751
synapse.lib.output
module, 752
synapse.lib.parser
module, 752
synapse.lib.platforms
module, 579
synapse.lib.platforms.common
module, 579
synapse.lib.platforms.darwin
module, 580
synapse.lib.platforms.freebsd
module, 580
synapse.lib.platforms.linux
module, 580
synapse.lib.platforms.windows
module, 581
synapse.lib.queue
module, 754
synapse.lib.ratelimit
module, 755
synapse.lib.reflect
module, 755
synapse.lib.rstorm
module, 756
synapse.lib.schemas
module, 757
synapse.lib.scope
module, 757
synapse.lib.scrape
module, 759
synapse.lib.share
module, 764
synapse.lib.slaboffs
module, 764
synapse.lib.slabseqn
module, 764
synapse.lib.snap
module, 766
synapse.lib.spooled
module, 770
synapse.lib.storm
module, 771
synapse.lib.storm_format
module, 788
synapse.lib.stormctrl
module, 788
synapse.lib.stormhttp
module, 789
synapse.lib.stormlib
module, 581
synapse.lib.stormlib.aha
module, 581
synapse.lib.stormlib.auth
module, 582
synapse.lib.stormlib.backup
module, 585
synapse.lib.stormlib.basex
module, 585
synapse.lib.stormlib.cache
module, 585
synapse.lib.stormlib.cell
module, 586
synapse.lib.stormlib.compression
module, 586
synapse.lib.stormlib.cortex
module, 587
synapse.lib.stormlib.easyperm
module, 591
synapse.lib.stormlib.ethereum
module, 591
synapse.lib.stormlib.gen
module, 591
synapse.lib.stormlib.gis
module, 592
synapse.lib.stormlib.graph
module, 592
synapse.lib.stormlib.hashes
module, 592
synapse.lib.stormlib.hex
module, 593
synapse.lib.stormlib.imap
module, 593
synapse.lib.stormlib.infosec
module, 594
synapse.lib.stormlib.ipv6
module, 595
synapse.lib.stormlib.iters
module, 595
synapse.lib.stormlib.json
module, 596
synapse.lib.stormlib.log
module, 596
synapse.lib.stormlib.macro
module, 597
synapse.lib.stormlib.math
module, 597
synapse.lib.stormlib.mime
module, 598
synapse.lib.stormlib.model

module, 598
synapse.lib.stormlib.modelext
module, 600
synapse.lib.stormlib.notifications
module, 600
synapse.lib.stormlib.oauth
module, 601
synapse.lib.stormlib.pack
module, 601
synapse.lib.stormlib.project
module, 602
synapse.lib.stormlib.random
module, 604
synapse.lib.stormlib.scrape
module, 604
synapse.lib.stormlib.smtp
module, 605
synapse.lib.stormlib.spooled
module, 605
synapse.lib.stormlib.stats
module, 606
synapse.lib.stormlib.stix
module, 607
synapse.lib.stormlib.storm
module, 608
synapse.lib.stormlib.vault
module, 609
synapse.lib.stormlib.version
module, 610
synapse.lib.stormlib.xml
module, 610
synapse.lib.stormlib.yaml
module, 610
synapse.lib.stormsvc
module, 790
synapse.lib.stormtypes
module, 790
synapse.lib.stormwhois
module, 809
synapse.lib.structlog
module, 809
synapse.lib.task
module, 809
synapse.lib.thishost
module, 810
synapse.lib.thisplat
module, 810
synapse.lib.threads
module, 810
synapse.lib.time
module, 811
synapse.lib.trigger
module, 812
synapse.lib.types
module, 813
synapse.lib.urlhelp
module, 820
synapse.lib.version
module, 821
synapse.lib.view
module, 823
synapse.lookup
module, 826
synapse.lookup.cvss
module, 826
synapse.lookup.iana
module, 826
synapse.lookup.iso3166
module, 826
synapse.lookup.macho
module, 827
synapse.lookup.pe
module, 827
synapse.lookup.phonenum
module, 827
synapse.lookup.timezones
module, 827
synapse.mindmeld
module, 939
synapse.models
module, 827
synapse.models.auth
module, 828
synapse.models.base
module, 828
synapse.models.belief
module, 828
synapse.models.biz
module, 829
synapse.models.crypto
module, 829
synapse.models.dns
module, 829
synapse.models.economic
module, 829
synapse.models.files
module, 829
synapse.models.geopol
module, 830
synapse.models.geospace
module, 830
synapse.models.gov
module, 827
synapse.models.gov.cn
module, 827
synapse.models.gov.intl
module, 828
synapse.models.gov.us

module, 828
synapse.models.inet
module, 831
synapse.models.infotech
module, 833
synapse.models.language
module, 834
synapse.models.material
module, 835
synapse.models.media
module, 835
synapse.models.orgs
module, 835
synapse.models.person
module, 835
synapse.models.proj
module, 835
synapse.models.risk
module, 836
synapse.models.science
module, 836
synapse.models.syn
module, 836
synapse.models.telco
module, 837
synapse.models.transport
module, 838
synapse.servers
module, 838
synapse.servers.aha
module, 838
synapse.servers.axon
module, 838
synapse.servers.cell
module, 838
synapse.servers.cortex
module, 838
synapse.servers.cryotank
module, 838
synapse.servers.jsonstor
module, 838
synapse.servers.stemcell
module, 838
synapse.telepath
module, 939
synapse.tests
module, 838
synapse.tests.nopmod
module, 838
synapse.tests.utils
module, 839
synapse.tools
module, 857
synapse.tools.aha
module, 857
synapse.tools.aha.easycert
module, 857
synapse.tools.aha.enroll
module, 857
synapse.tools.aha.list
module, 857
synapse.tools.aha.provision
module, 857
synapse.tools.aha.provision.service
module, 857
synapse.tools.aha.provision.user
module, 857
synapse.tools.autodoc
module, 858
synapse.tools.axon2axon
module, 860
synapse.tools.backup
module, 860
synapse.tools.cellauth
module, 861
synapse.tools.cmdr
module, 861
synapse.tools.cryo
module, 857
synapse.tools.cryo.cat
module, 857
synapse.tools.cryo.list
module, 858
synapse.tools.csvtool
module, 861
synapse.tools.docker
module, 858
synapse.tools.docker.validate
module, 858
synapse.tools.easycert
module, 861
synapse.tools.feed
module, 861
synapse.tools.genpkg
module, 862
synapse.tools.guid
module, 862
synapse.tools.healthcheck
module, 863
synapse.tools.hive
module, 858
synapse.tools.hive.load
module, 858
synapse.tools.hive.save
module, 858
synapse.tools.json2mpk
module, 863
synapse.tools.livebackup

module, 863
synapse.tools.modrole
module, 863
synapse.tools.moduser
module, 863
synapse.tools.promote
module, 863
synapse.tools.pullfile
module, 863
synapse.tools.pushfile
module, 864
synapse.tools.reload
module, 864
synapse.tools.rstorm
module, 864
synapse.tools.storm
module, 864
synapse.utils
module, 867
synapse.utils.getrefs
module, 871
synapse.utils.stormcov
module, 867
synapse.utils.stormcov.plugin
module, 868
sync() (*in module synapse.glob*), 938
sync() (*synapse.lib.cell.Cell method*), 655
sync() (*synapse.lib.lmdbslab.HotKeyVal method*), 730
sync() (*synapse.lib.lmdbslab.Slab method*), 735
syncevnt (*synapse.lib.lmdbslab.Slab attribute*), 735
synchelp() (*in module synapse.glob*), 938
syncIndexEvents() (*synapse.cortex.CoreApi method*), 901
syncIndexEvents() (*synapse.cortex.Cortex method*), 920
syncIndexEvents() (*synapse.lib.layer.Layer method*), 724
syncLayerNodeEdits() (*synapse.cortex.CoreApi method*), 901
syncLayerNodeEdits() (*synapse.cortex.Cortex method*), 921
syncLayersEvents() (*synapse.cortex.CoreApi method*), 901
syncLayersEvents() (*synapse.cortex.Cortex method*), 921
syncLoopOnce() (*synapse.lib.lmdbslab.Slab class method*), 735
syncLoopTask() (*synapse.lib.lmdbslab.Slab class method*), 735
syncNodeEdits() (*synapse.lib.layer.Layer method*), 725
syncNodeEdits() (*synapse.lib.layer.LayerApi method*), 726
syncNodeEdits2() (*synapse.lib.layer.Layer method*), 725
syncNodeEdits2() (*synapse.lib.layer.LayerApi method*), 726
syncNodeEdits2() (*synapse.lib.view.ViewApi method*), 826
synctask (*synapse.lib.lmdbslab.Slab attribute*), 735
SynErr, 937
SynModule (*class in synapse.models.syn*), 836
SynTest (*class in synapse.tests.utils*), 841

T

t2call() (*in module synapse.daemon*), 925
Tag (*class in synapse.lib.types*), 817
tag() (*in module synapse.lib.chop*), 679
tagcachesize (*synapse.lib.snap.Snap attribute*), 769
TagCond (*class in synapse.lib.ast*), 631
tagged() (*in module synapse.lib.node*), 750
TagGlobs (*class in synapse.lib.cache*), 642
TagMatch (*class in synapse.lib.ast*), 631
TagMatchRe (*in module synapse.lib.chop*), 677
TagName (*class in synapse.lib.ast*), 631
TagPart (*class in synapse.lib.types*), 817
tagpath() (*in module synapse.lib.chop*), 679
TagProp (*class in synapse.datamodel*), 929
TagProp (*class in synapse.lib.ast*), 631
tagprop() (*synapse.datamodel.Model method*), 928
TagPropCond (*class in synapse.lib.ast*), 631
tagpropreprs() (*synapse.lib.node.Node method*), 747
TagPropValue (*class in synapse.lib.ast*), 631
TagPruneCmd (*class in synapse.lib.storm*), 786
tags() (*in module synapse.lib.chop*), 679
tags() (*in module synapse.lib.node*), 751
tagsnice() (*in module synapse.lib.node*), 751
TagValuCond (*class in synapse.lib.ast*), 631
TagValue (*class in synapse.lib.ast*), 631
tally() (*synapse.lib.stormlib.stats.LibStats method*), 606
TankApi (*class in synapse.cryotank*), 924
tankapi (*synapse.cryotank.CryoCell attribute*), 923
Task (*class in synapse.lib.task*), 809
Task (*class in synapse.telepath*), 941
task() (*synapse.telepath.Client method*), 939
task() (*synapse.telepath.Proxy method*), 941
taskv2() (*synapse.telepath.Proxy method*), 941
Taxon (*class in synapse.lib.types*), 817
Taxonomy (*class in synapse.lib.types*), 817
TeeCmd (*class in synapse.lib.storm*), 786
TelcoModule (*class in synapse.models.telco*), 837
TeleHive (*class in synapse.lib.hive*), 697
TeleSSLObject (*class in synapse.telepath*), 941
TestCmd (*class in synapse.tests.utils*), 854
testguid (*synapse.tests.utils.TestModule attribute*), 855
TestModule (*class in synapse.tests.utils*), 854

TestRunt (*class in synapse.tests.utils*), 855
TestSubType (*class in synapse.tests.utils*), 855
TestType (*class in synapse.tests.utils*), 855
Text (*class in synapse.lib.stormtypes*), 806
text (*synapse.lib.parser.AstInfo attribute*), 753
textFromRule() (*in module synapse.lib.hiveauth*), 701
textFromRule() (*synapse.lib.stormlib.auth.LibAuth method*), 582
thisHostMust() (*synapse.tests.utils.SynTest method*), 852
thisHostMustNot() (*synapse.tests.utils.SynTest method*), 852
ThreeType (*class in synapse.tests.utils*), 855
tick() (*synapse.lib.storm.Runtime method*), 784
Time (*class in synapse.lib.types*), 817
TimeEdge (*class in synapse.lib.types*), 818
TimeOut, 938
timestamp() (*synapse.lib.stormlib.stix.LibStixExport method*), 607
TimeUnit (*class in synapse.lib.agenda*), 612
timewait() (*synapse.lib.coro.Event method*), 687
TinFoilHat (*class in synapse.lib.crypto.tinfoil*), 578
toaxon() (*synapse.lib.stormtypes.LibExport method*), 795
tobool() (*in module synapse.lib.stormtypes*), 808
tobuidhex() (*in module synapse.lib.stormtypes*), 808
tocmprvalu() (*in module synapse.lib.stormtypes*), 808
todo() (*in module synapse.common*), 894
toint() (*in module synapse.lib.stormtypes*), 808
toint() (*synapse.lib.stormlib.hex.HexLib method*), 593
toiter() (*in module synapse.lib.stormtypes*), 808
tonumber() (*in module synapse.lib.stormtypes*), 808
toprim() (*in module synapse.lib.stormtypes*), 808
torepr() (*in module synapse.lib.stormtypes*), 808
tostor() (*in module synapse.lib.stormtypes*), 808
tostr() (*in module synapse.lib.stormtypes*), 808
totext() (*synapse.lib.stormlib.mime.LibMimeHtml method*), 598
totype() (*in module synapse.lib.stormtypes*), 808
toUTC() (*in module synapse.lib.time*), 812
toUTC() (*synapse.lib.stormtypes.LibTime method*), 799
TransportModule (*class in synapse.models.transport*), 838
trash() (*synapse.lib.lmdbslab.Slab method*), 735
treeAndSync() (*synapse.lib.hive.HiveApi method*), 695
TreeCmd (*class in synapse.lib.storm*), 787
Trigger (*class in synapse.lib.stormtypes*), 806
Trigger (*class in synapse.lib.trigger*), 812
Triggers (*class in synapse.lib.trigger*), 812
trim() (*synapse.lib.slabseqn.SlabSeqn method*), 766
trimext() (*synapse.lib.stormlib.hex.HexLib method*), 593
trimNexsLog() (*synapse.lib.cell.Cell method*), 655
trimNexsLog() (*synapse.lib.cell.CellApi method*), 661
true() (*synapse.tests.utils.SynTest method*), 853
trycast() (*synapse.lib.stormtypes.LibBase method*), 793
TryCatch (*class in synapse.lib.ast*), 632
tryDynFunc() (*in module synapse.lib.dyndeps*), 690
tryDynLocal() (*in module synapse.lib.dyndeps*), 690
tryDynMod() (*in module synapse.lib.dyndeps*), 690
tryLoadPkgProto() (*in module synapse.tools.genpkg*), 862
tryPasswd() (*synapse.lib.hiveauth.HiveUser method*), 701
tryToMerge() (*synapse.lib.view.View method*), 826
tryUserPasswd() (*synapse.lib.cell.Cell method*), 655
tryUserPasswd() (*synapse.lib.cell.CellApi method*), 661
TstEnv (*class in synapse.tests.utils*), 856
TstOutPut (*class in synapse.tests.utils*), 856
tuplify() (*in module synapse.common*), 894
tx() (*synapse.lib.link.Link method*), 728
tx() (*synapse.lib.stormhttp.WebSocket method*), 790
txfini() (*synapse.lib.link.Link method*), 728
txnbackup() (*in module synapse.tools.backup*), 860
Type (*class in synapse.lib.types*), 818
type() (*synapse.datamodel.Model method*), 928
typeerr() (*in module synapse.lib.stormtypes*), 808
typename (*synapse.axon.UpLoadShare attribute*), 886
typename (*synapse.daemon.AsyncGenr attribute*), 925
typename (*synapse.daemon.Genr attribute*), 925

U

uhex() (*in module synapse.common*), 894
un() (*in module synapse.lib.msgpack*), 741
un() (*synapse.lib.stormlib.compression.Bzip2Lib method*), 586
un() (*synapse.lib.stormlib.compression.GzipLib method*), 587
un() (*synapse.lib.stormlib.compression.ZlibLib method*), 587
un() (*synapse.lib.stormlib.pack.LibPack method*), 602
UnaryExprNode (*class in synapse.lib.ast*), 632
unc_path_check() (*in module synapse.lib.scrape*), 763
uncnorm() (*in module synapse.lib.chop*), 679
Undef (*class in synapse.lib.stormtypes*), 807
undefined_types (*synapse.lib.stormtypes.StormTypesRegistry attribute*), 806
unescape() (*in module synapse.lib.parser*), 754
UniqCmd (*class in synapse.lib.storm*), 787
univ() (*synapse.datamodel.Model method*), 928
UnivProp (*class in synapse.lib.ast*), 632
UnivPropValue (*class in synapse.lib.ast*), 632
unixconnect() (*in module synapse.lib.link*), 729
unixlisten() (*in module synapse.lib.link*), 729
unjsonsafe_nodeedits() (*in module synapse.common*), 895

unlink() (*synapse.lib.base.Base method*), 638
unpack() (*synapse.lib.agenda.ApptRec class method*), 612
unpackVersion() (*in module synapse.lib.version*), 822
Unpk (*class in synapse.lib.msgpack*), 740
update() (*in module synapse.lib.scope*), 759
update() (*synapse.lib.hashset.HashSet method*), 693
update() (*synapse.lib.health.HealthCheck method*), 693
update() (*synapse.lib.httpapi.Sess method*), 708
update() (*synapse.lib.nexus.ChangeDist method*), 743
update() (*synapse.lib.scope.Scope method*), 758
updateCronJob() (*synapse.cortex.Cortex method*), 921
updateHttpSessInfo() (*synapse.lib.cell.Cell method*), 655
UpLoad (*class in synapse.axon*), 886
upload() (*synapse.axon.Axon method*), 876
upload() (*synapse.axon.AxonApi method*), 882
upload() (*synapse.lib.stormtypes.LibAxon method*), 793
UpLoadProxy (*class in synapse.axon*), 886
UpLoadShare (*class in synapse.axon*), 886
Url (*class in synapse.models.inet*), 832
urldecode() (*synapse.lib.stormhttp.LibHttp method*), 789
urlencode() (*synapse.lib.stormhttp.LibHttp method*), 789
urlfile() (*synapse.lib.stormtypes.LibAxon method*), 793
User (*class in synapse.lib.stormlib.auth*), 583
user() (*in module synapse.lib.task*), 809
user() (*synapse.lib.hiveauth.Auth method*), 700
useriden() (*synapse.lib.httpapi.HandlerBase method*), 707
UserJson (*class in synapse.lib.stormlib.auth*), 584
username() (*in module synapse.lib.task*), 809
UserProfile (*class in synapse.lib.stormlib.auth*), 584
users() (*synapse.lib.hiveauth.Auth method*), 700
UserVars (*class in synapse.lib.stormlib.auth*), 584
uuid4() (*in module synapse.lib.stormlib.stix*), 608
uuid5() (*in module synapse.lib.stormlib.stix*), 608

V

valCodeCert() (*synapse.lib.certdir.CertDir method*), 676
validate() (*synapse.lib.ast.ArgvQuery method*), 618
validate() (*synapse.lib.ast.AstNode method*), 618
validate() (*synapse.lib.ast.EmbedQuery method*), 621
validate() (*synapse.lib.ast.Function method*), 624
validate() (*synapse.lib.ast.VarValue method*), 633
validateBundle() (*synapse.lib.stormlib.stix.LibStix method*), 607
validateStix() (*in module synapse.lib.stormlib.stix*), 608
validateTagMatch() (*in module synapse.lib.chop*), 679
validedgekeys (*synapse.lib.stormlib.model.LibModelEdge attribute*), 599
vals() (*synapse.lib.base.BaseRef method*), 640
valu() (*synapse.lib.parser.CmdStringer method*), 753
Value (*class in synapse.lib.ast*), 632
value() (*synapse.lib.ast.Const method*), 619
value() (*synapse.lib.stormlib.auth.Role method*), 583
value() (*synapse.lib.stormlib.auth.User method*), 584
value() (*synapse.lib.stormlib.auth.UserProfile method*), 584
value() (*synapse.lib.stormlib.model.ModelForm method*), 599
value() (*synapse.lib.stormlib.model.ModelProp method*), 599
value() (*synapse.lib.stormlib.model.ModelTagProp method*), 599
value() (*synapse.lib.stormlib.model.ModelType method*), 600
value() (*synapse.lib.stormlib.project.Project method*), 602
value() (*synapse.lib.stormlib.project.ProjectEpic method*), 602
value() (*synapse.lib.stormlib.project.ProjectSprint method*), 603
value() (*synapse.lib.stormlib.project.ProjectTicket method*), 603
value() (*synapse.lib.stormlib.project.ProjectTicketComment method*), 603
value() (*synapse.lib.stormlib.spooled.SpooledSet method*), 605
value() (*synapse.lib.stormlib.stats.StatTally method*), 606
value() (*synapse.lib.stormlib.stix.StixBundle method*), 608
value() (*synapse.lib.stormlib.vault.Vault method*), 609
value() (*synapse.lib.stormlib.vault.VaultConfigs method*), 609
value() (*synapse.lib.stormtypes.CmdOpts method*), 791
value() (*synapse.lib.stormtypes.Dict method*), 791
value() (*synapse.lib.stormtypes.List method*), 800
value() (*synapse.lib.stormtypes.NodeProps method*), 801
value() (*synapse.lib.stormtypes.Prim method*), 802
value() (*synapse.lib.stormtypes.StormHiveDict method*), 805
values() (*synapse.lib.cache.LruDict method*), 642
values() (*synapse.lib.hive.HiveDict method*), 696
valUserCert() (*synapse.lib.certdir.CertDir method*), 676
vardefault() (*in module synapse.lib.task*), 810
VarDeref (*class in synapse.lib.ast*), 632
varderef() (*synapse.lib.parser.AstConverter method*), 753
VarEvalOper (*class in synapse.lib.ast*), 632

varget() (*in module synapse.lib.task*), 810
varinit() (*in module synapse.lib.task*), 810
VarList (*class in synapse.lib.ast*), 632
varlist() (*synapse.lib.parser.AstConverter method*), 753
VarListSetOper (*class in synapse.lib.ast*), 632
varset() (*in module synapse.lib.task*), 810
VarValue (*class in synapse.lib.ast*), 633
Vault (*class in synapse.lib.stormlib.vault*), 609
VaultConfigs (*class in synapse.lib.stormlib.vault*), 609
VaultSecrets (*class in synapse.lib.stormlib.vault*), 609
vcr (*synapse.tests.utils.StormPkgTest attribute*), 841
vectToProps() (*synapse.lib.stormlib.infosec.CvssLib method*), 595
vectToScore() (*synapse.lib.stormlib.infosec.CvssLib method*), 595
Velocity (*class in synapse.lib.types*), 820
verify() (*synapse.lib.crypto.ecc.PubKey method*), 574
verify() (*synapse.lib.crypto.rsa.PubKey method*), 577
verify() (*synapse.lib.layer.Layer method*), 725
verify() (*synapse.lib.stormtypes.Layer method*), 792
verifyAllBuids() (*synapse.lib.layer.Layer method*), 725
verifyAllProps() (*synapse.lib.layer.Layer method*), 725
verifyAllTagProps() (*synapse.lib.layer.Layer method*), 725
verifyAllTags() (*synapse.lib.layer.Layer method*), 725
verifyBuidProp() (*synapse.lib.layer.StorType method*), 726
verifyBuidTag() (*synapse.lib.layer.Layer method*), 725
verifyByBuid() (*synapse.lib.layer.Layer method*), 725
verifyByProp() (*synapse.lib.layer.Layer method*), 725
verifyByPropArray() (*synapse.lib.layer.Layer method*), 725
verifyByTag() (*synapse.lib.layer.Layer method*), 725
verifyByTagProp() (*synapse.lib.layer.Layer method*), 725
verifyitem() (*synapse.lib.crypto.rsa.PubKey method*), 577
verifyPbkdf2() (*in module synapse.lib.crypto.passwd*), 576
verifyStormPkgDeps() (*synapse.cortex.Cortex method*), 921
VERSION (*synapse.lib.cell.Cell attribute*), 643
VersionLib (*class in synapse.lib.stormlib.version*), 610
verstr() (*in module synapse.common*), 895
VERSTRING (*synapse.lib.cell.Cell attribute*), 643
vertup() (*in module synapse.common*), 895
View (*class in synapse.lib.stormtypes*), 807
View (*class in synapse.lib.view*), 823
ViewApi (*class in synapse.lib.view*), 826
viewapi (*synapse.cortex.Cortex attribute*), 921
viewctor() (*synapse.cortex.Cortex class method*), 921
viewDynCall() (*synapse.lib.stormtypes.View method*), 807
viewDynIter() (*synapse.lib.stormtypes.View method*), 807
ViewExecCmd (*class in synapse.lib.storm*), 787

W

wait() (*synapse.lib.base.Waiter method*), 640
wait() (*synapse.tests.utils.AsyncStreamEvent method*), 839
wait_for() (*in module synapse.common*), 895
waitEditOffs() (*synapse.lib.layer.Layer method*), 725
Waiter (*class in synapse.lib.base*), 640
waiter() (*synapse.lib.base.Base method*), 639
waitfini() (*synapse.lib.base.Base method*), 639
waitForHot() (*synapse.lib.layer.Layer method*), 725
waitForOffset() (*synapse.lib.multislabseqn.MultiSlabSeqn method*), 743
waitForOffset() (*synapse.lib.slabseqn.SlabSeqn method*), 766
waitNexsOffs() (*synapse.lib.cell.Cell method*), 655
waitNexsOffs() (*synapse.lib.cell.CellApi method*), 661
waitOffs() (*synapse.lib.nexus.NexsRoot method*), 744
waitready() (*synapse.telepath.Client method*), 939
waitready() (*synapse.telepath.ClientV2 method*), 940
waitStormSvc() (*synapse.cortex.Cortex method*), 921
waittask() (*in module synapse.lib.coro*), 689
waitUpstreamOffs() (*synapse.lib.layer.Layer method*), 725
walkNodeEdges() (*synapse.lib.ast.N1Walk method*), 626
walkNodeEdges() (*synapse.lib.ast.N2Walk method*), 626
wants() (*synapse.axon.Axon method*), 876
wants() (*synapse.axon.AxonApi method*), 883
warn() (*synapse.lib.snap.Snap method*), 769
warn() (*synapse.lib.storm.Runtime method*), 784
WARN_COMMIT_TIME_MS (*synapse.lib.lmdbslab.Slab attribute*), 732
warnonce() (*synapse.lib.snap.Snap method*), 770
warnonce() (*synapse.lib.storm.Runtime method*), 784
wasAdded() (*synapse.datamodel.Form method*), 926
wasDel() (*synapse.datamodel.Prop method*), 929
wasDeleted() (*synapse.datamodel.Form method*), 927
wasSet() (*synapse.datamodel.Prop method*), 929
watchAllUserNotifs() (*synapse.cortex.CoreApi method*), 901
watchAllUserNotifs() (*synapse.cortex.Cortex method*), 921
watchAllUserNotifs() (*synapse.lib.jsonstor.JsonStorApi method*), 712

watchAllUserNotifs() (*synapse.lib.jsonstor.JsonStorCell method*), 713
WebSocket (*class in synapse.lib.httpapi*), 710
WebSocket (*class in synapse.lib.stormhttp*), 789
wget() (*synapse.axon.Axon method*), 877
wget() (*synapse.axon.AxonApi method*), 883
wget() (*synapse.lib.stormtypes.LibAxon method*), 793
WhileLoop (*class in synapse.lib.ast*), 633
wildrange() (*in module synapse.lib.time*), 812
Window (*class in synapse.lib.queue*), 755
windows_path_check() (*in module synapse.lib.scrape*), 763
wipeAllowed() (*synapse.lib.view.View method*), 826
wipeLayer() (*synapse.lib.view.View method*), 826
withCliPromptMock() (*synapse.tests.utils.SynTest method*), 853
withCliPromptMockExtendOutp() (*synapse.tests.utils.SynTest method*), 853
withNexusReplay() (*synapse.tests.utils.SynTest method*), 853
withSetLoggingMock() (*synapse.tests.utils.SynTest method*), 853
withStableUids() (*synapse.tests.utils.SynTest method*), 854
withTeleEnv() (*in module synapse.telepath*), 942
withTestCmdr() (*synapse.tests.utils.SynTest method*), 854
worker() (*in module synapse.common*), 895
worker() (*synapse.lib.task.Task method*), 809
wput() (*synapse.axon.Axon method*), 878
wput() (*synapse.axon.AxonApi method*), 884
wput() (*synapse.lib.stormtypes.LibAxon method*), 793
wrap_liftgenr() (*in module synapse.cortex*), 922
write() (*synapse.axon.UpLoad method*), 886
write() (*synapse.axon.UpLoadProxy method*), 886
write() (*synapse.tests.utils.AsyncStreamEvent method*), 839
write() (*synapse.tests.utils.StreamEvent method*), 841

X

xmit() (*synapse.lib.httpapi.WebSocket method*), 710
XmlElement (*class in synapse.lib.stormlib.xml*), 610
xrp_check() (*in module synapse.lib.crypto.coin*), 572

Y

yamlload() (*in module synapse.common*), 895
yamlloads() (*in module synapse.common*), 895
yamlmod() (*in module synapse.common*), 895
yamlpop() (*in module synapse.common*), 895
yamlsave() (*in module synapse.common*), 895
YEAR (*synapse.lib.agenda.TimeUnit attribute*), 612
year() (*in module synapse.lib.time*), 812
year() (*synapse.lib.stormtypes.LibTime method*), 799
yieldFromValu() (*synapse.lib.ast.YieldValu method*), 633
YieldValu (*class in synapse.lib.ast*), 633
yieldvalu() (*synapse.lib.parser.AstConverter method*), 753

Z

zipCpe22() (*in module synapse.models.infotech*), 834
zipurl() (*in module synapse.telepath*), 943
ZlibLib (*class in synapse.lib.stormlib.compression*), 587