

Synapse Bootcamp - Module 7

Pre-Storm Background - Exercises

Pre-Storm Background - Exercises	1
Objectives	1
Exercises	3
Form Categories	3
Exercise 1	3
Form and Property Namespaces	6
Exercise 2	6
Exercise 3	8
Exercise 4	9
Type Enforcement	11
Exercise 5	11
Type-Specific Behavior	18
Exercise 6	18
Type Awareness	20
Exercise 7	20

Objectives

In these exercises you will learn:

- How to examine the data model in greater detail
- How to identify what a form represents (e.g., an object, a relationship, or an event)
- How to specify a full vs. a relative property
- How type enforcement normalizes data and helps prevent "bad data" from getting in to Synapse
- How type-specific behavior can make your Synapse and Storm life easier

Note: We are constantly updating Synapse and its Power-Ups! We do our best to make sure our course documents (slides, exercises, and answer keys) are up-to-date. However, you may notice small differences (such as between a screen capture in the documents and the appearance of your current instance of Synapse).

If something is unclear or if you identify an error, please reach out to us so we can assist!

Exercises

Form Categories

Exercise 1

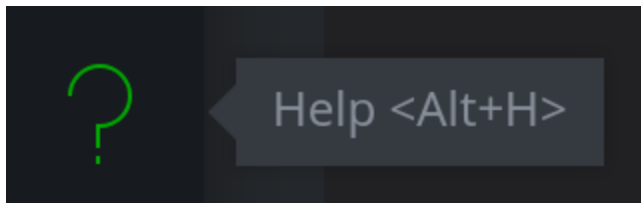
Objectives:

- Use Data Model Explorer to examine the data model in more detail.
- Determine "what" is being modeled and understand what a form represents.

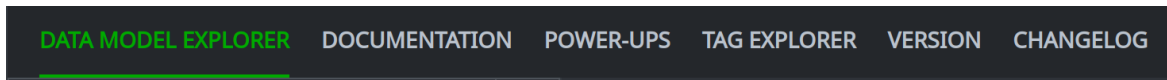
Part 1

View the **file:path** form and understand what it represents.

- From your **Toolbar**, select the **Help Tool**:



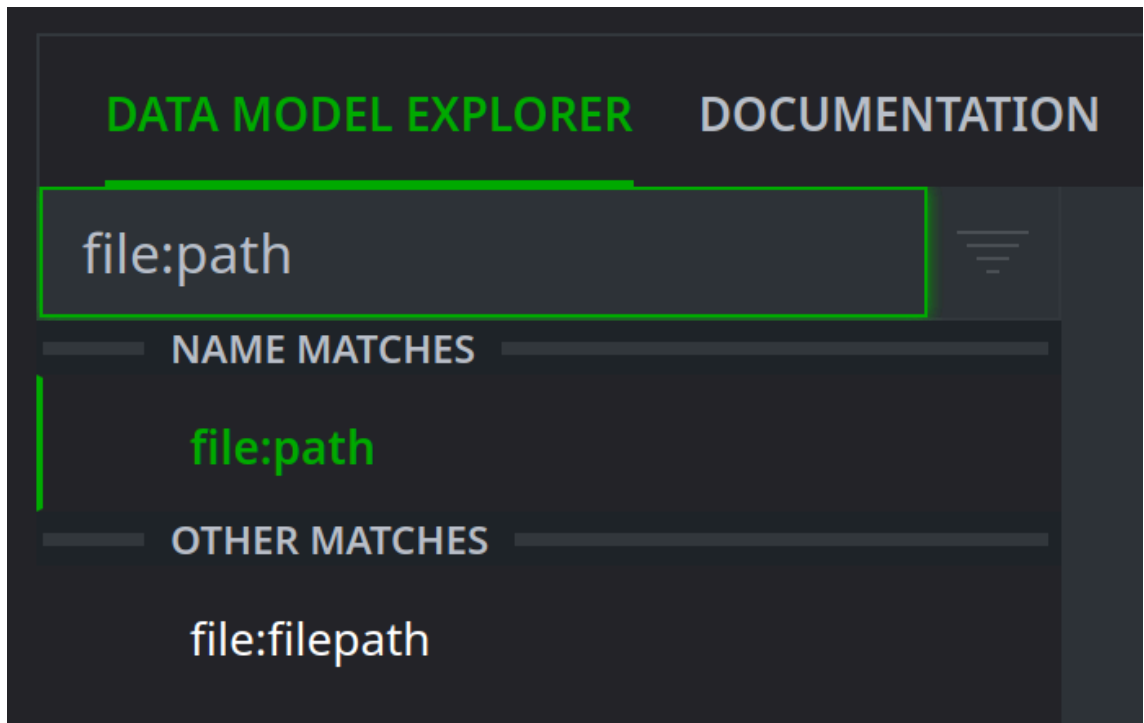
- In the **Help Tool**, click the **Data Model Explorer** tab:



- Enter the following in the *Search* field:

file:path

- Review the data model information for the **file:path** form:



Question 1: What does the **file:path** form represent?

Question 2: What is the form's primary property value? What would an example look like?

Question 3: Is a **file:path** an object, a relationship, or an event?

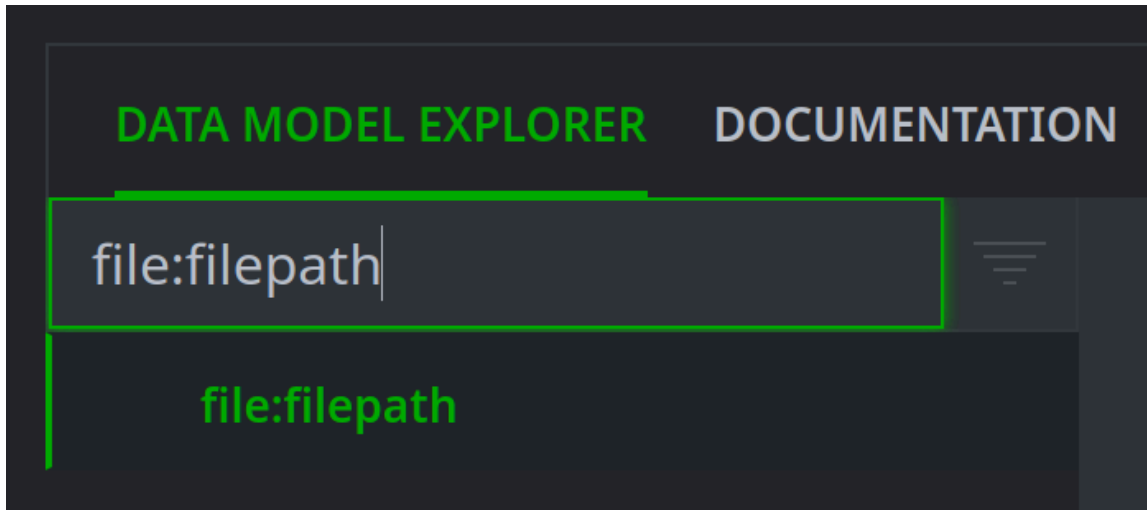
Part 2

View the **file:filepath** form and understand what it represents.

- In **Data Model Explorer**, enter the following in the *Search* field:

file:filepath

- Review the data model information for the **file:filepath** form:



Question 4: What does the **file:filepath** form represent?

Question 5: What is the form's primary property value? What would an example look like?

Question 6: Is a **file:filepath** an object, a relationship, or an event?

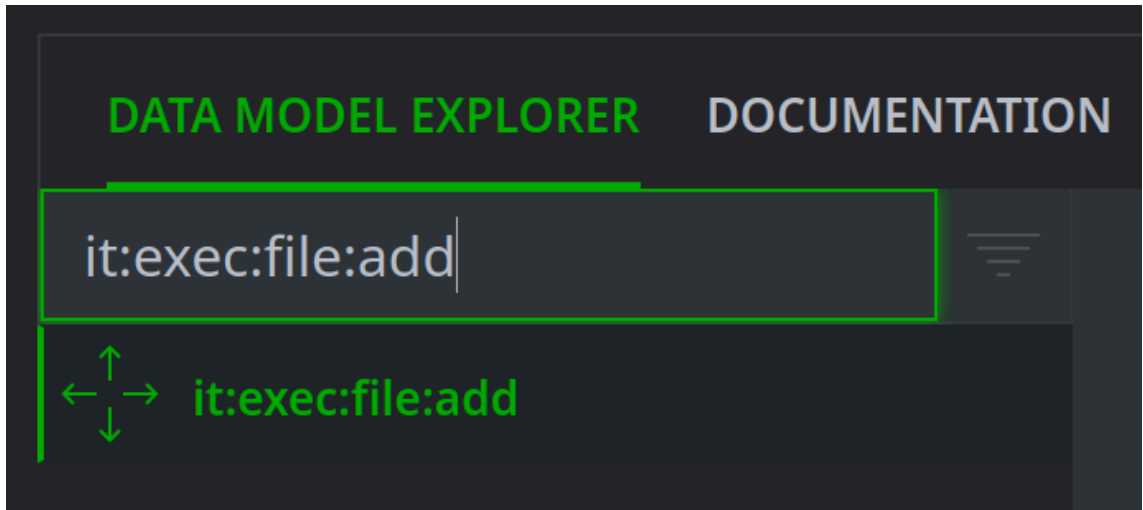
Part 3

View the **it:exec:file:add** form and understand what it represents.

- In **Data Model Explorer**, enter the following in the *Search* field:

it:exec:file:add

- Review the data model information for the **it:exec:file:add** form:



Question 7: What does the **it:exec:file:add** form represent?

Question 8: What is the form's primary property value? What would an example look like?

Question 9: Is an **it:exec:file:add** form an object, a relationship, or an event?

Form and Property Namespaces

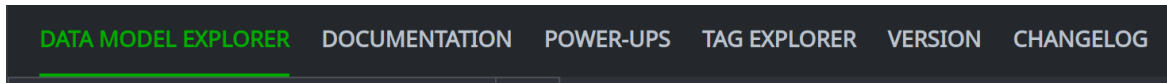
Exercise 2

Objectives:

- Use the Data Model Explorer to view the subset of forms within a particular namespace.
- See examples of how forms in the data model may be grouped together.
- Understand when a "subcategory" may be used to group related forms within a larger namespace.

View forms in the **inet:** (Internet) category. Understand the kinds of objects in this part of the data model.

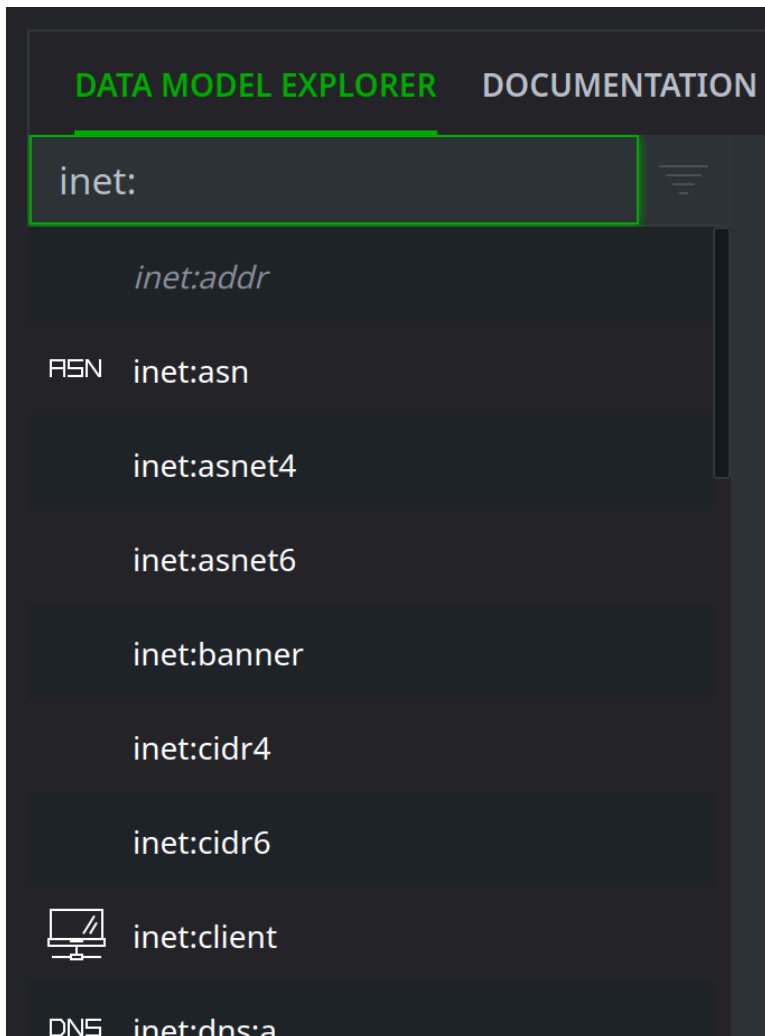
- In the **Help Tool**, click the **Data Model Explorer** tab:



- Enter the following in the *Search* field:

inet:

- Using the list view, browse the various form names that start with **inet:** (i.e., forms in the **inet** or "Internet" category):



Question 1: What types of things (forms, objects) are in the **inet:** category?

Question 2: What "subcategories" can you identify in the **inet:** category?

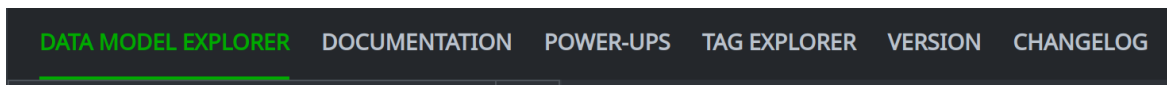
Exercise 3

Objectives:

- Understand the difference between a form name / namespace and a property name / namespace.
- Identify both the full and relative names of a property.

View the **inet:whois:rec** form and identify its properties.

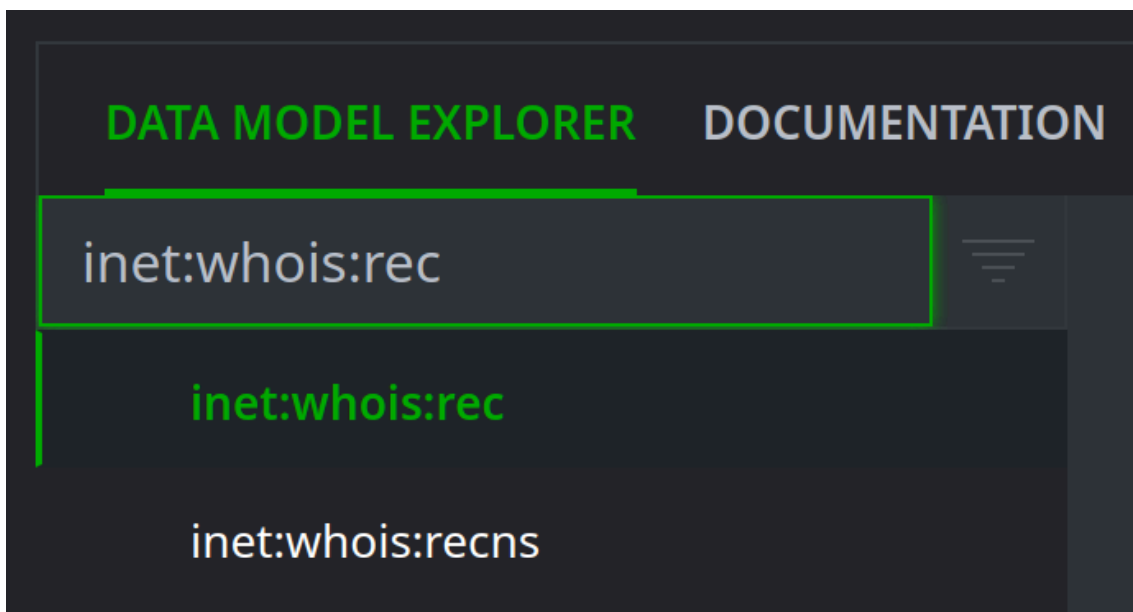
- In the **Help Tool**, click the **Data Model Explorer** tab:



- Enter the following in the *Search* field:

inet:whois:rec

- Select **inet:whois:rec** from the list view:



- **Review** the data model information for this form:

inet:whois:rec

[Lift in Research Tool](#)
[docs link](#)

A domain whois record.

type: inet:whois:rec
base: comp
(fqdn[inet:fqdn], asof[time])

Properties

name	ro	type	doc
:fqdn	—	inet:fqdn	The domain associated with the whois record.
:asof	—	time	The date of the whois record.
:created		time	The "created" time from the whois record.
:expires		time	The "expires" time from the whois record.
:registrant		inet:whois:reg	The registrant name from the whois record.
:registrar		inet:whois:rar	The registrar name from the whois record.
:text		str	The full text of the whois record.
:updated		time	The "last updated" time from the whois record.
.created	—	time	The time the node was created in the cortex.
.seen		ival	The time interval for first/last observation of the node.

- Locate the property of the **inet:whois:rec** form that represents the name of the person or organization that registered the domain (the "registrant").

Question 1: What is the **full property name** of the "registrant" property?

Question 2: What is the **relative property name** of the "registrant" property?

Exercise 4

Objectives:

- Understand the difference between a form name / namespace and a property name / namespace.
- Identify both the full and relative names of a property.

View the **file:bytes** form and identify its properties.

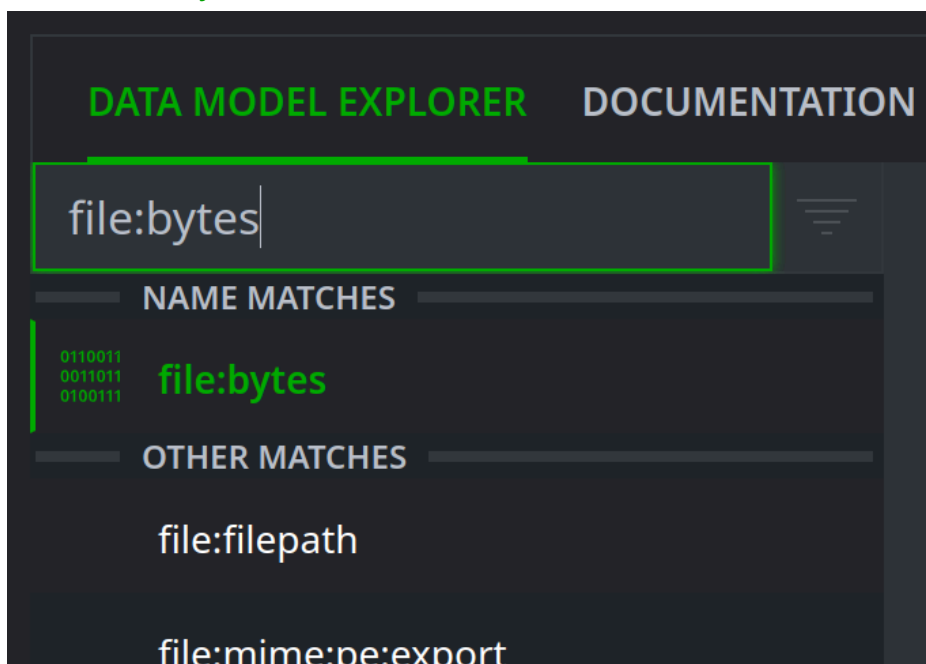
- In the **Help Tool**, click the **Data Model Explorer** tab:

DATA MODEL EXPLORER
DOCUMENTATION
POWER-UPS
TAG EXPLORER
VERSION
CHANGELOG

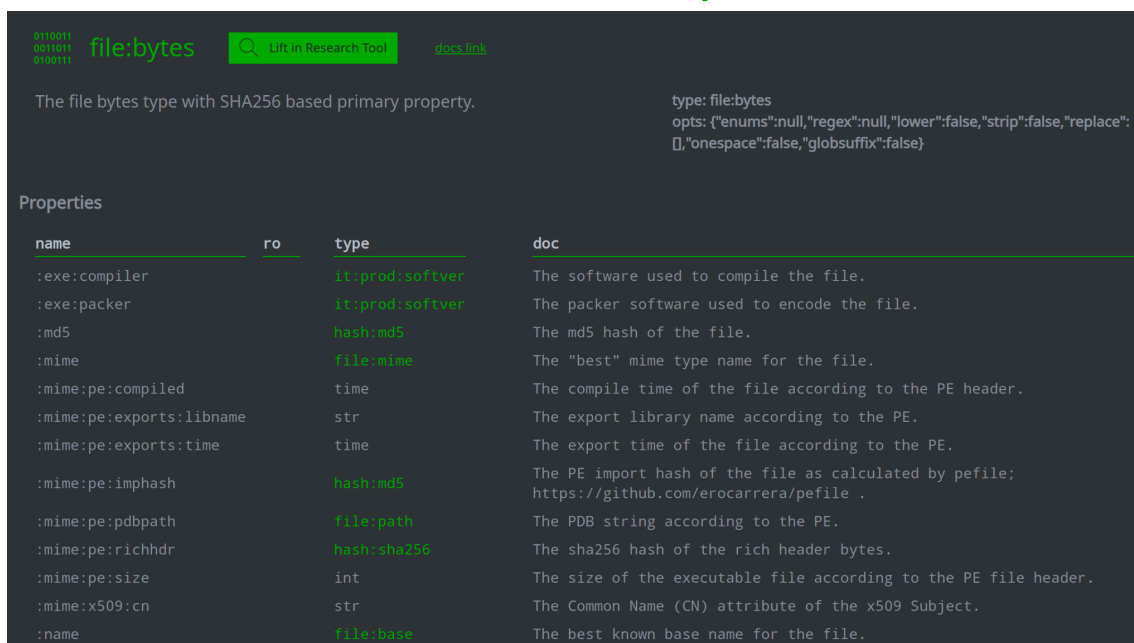
- Enter the following in the *Search* field:

file:bytes

- Select **file:bytes** from the list view:



- Review the data model information for the **file:bytes** form:



The screenshot shows the data model information for the 'file:bytes' form. At the top, there is a header with the text 'file:bytes' and a search icon. Below the header, there is a description: 'The file bytes type with SHA256 based primary property.' To the right of the description, there is a 'docs link' button. Below the description, there is a table with the following columns: 'name', 'ro', 'type', and 'doc'.

name	ro	type	doc
:exe:compiler		it:prod:softver	The software used to compile the file.
:exe:packer		it:prod:softver	The packer software used to encode the file.
:md5		hash:md5	The md5 hash of the file.
:mime		file:mime	The "best" mime type name for the file.
:mime:pe:compiled		time	The compile time of the file according to the PE header.
:mime:pe:exports:libname		str	The export library name according to the PE.
:mime:pe:exports:time		time	The export time of the file according to the PE.
:mime:pe:imphash		hash:md5	The PE import hash of the file as calculated by pefile; https://github.com/erocarrera/pefile .
:mime:pe:pdopath		file:path	The PDB string according to the PE.
:mime:pe:richhdr		hash:sha256	The sha256 hash of the rich header bytes.
:mime:pe:size		int	The size of the executable file according to the PE file header.
:mime:x509:cn		str	The Common Name (CN) attribute of the x509 Subject.
:name		file:base	The best known base name for the file.

Question 1: What is the **full property name** of the PE import hash (imphash) property?

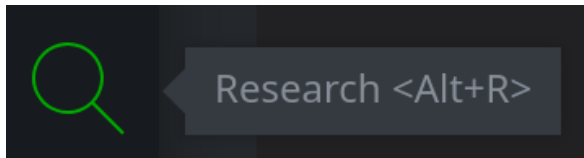
Question 2: What is the **relative property name** of this property?

Type Enforcement

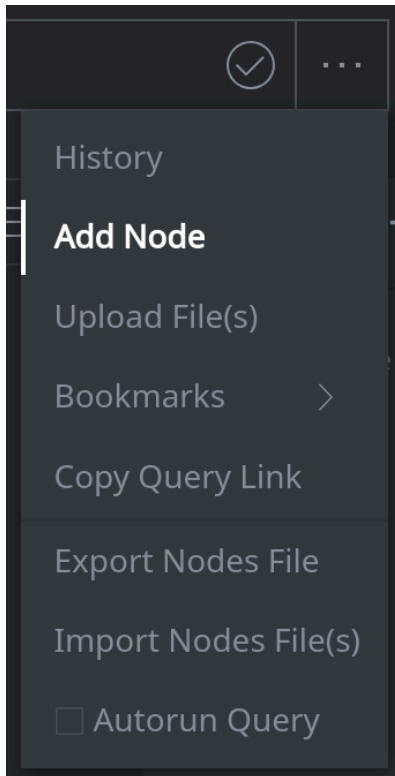
Exercise 5

Objective:

- Observe how Synapse helps to ensure data is consistent and correct through normalization and type enforcement.
-
- From your **Toolbar**, select the **Research Tool**:



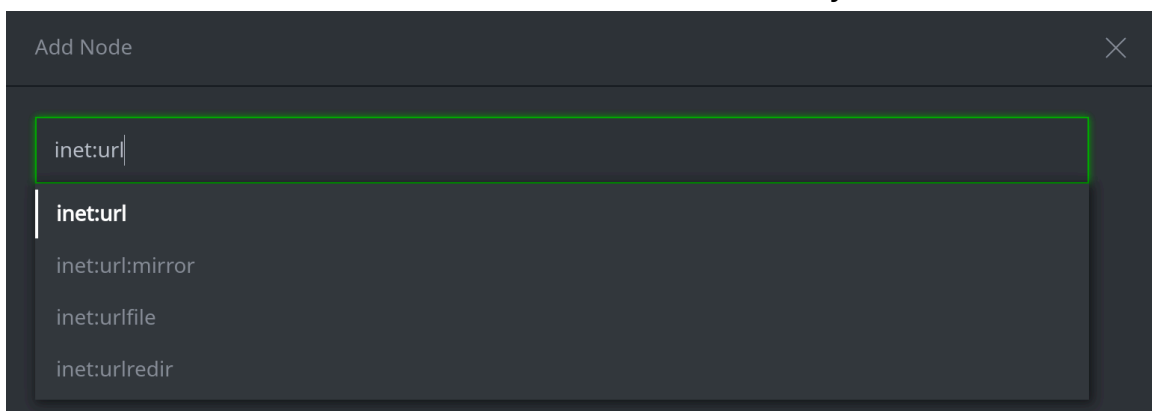
- Click the **Storm Query Bar** menu and choose **Add Node** from the dropdown list:



- In the **Add Node** dialog, enter the following in the *Form* field:

inet:url

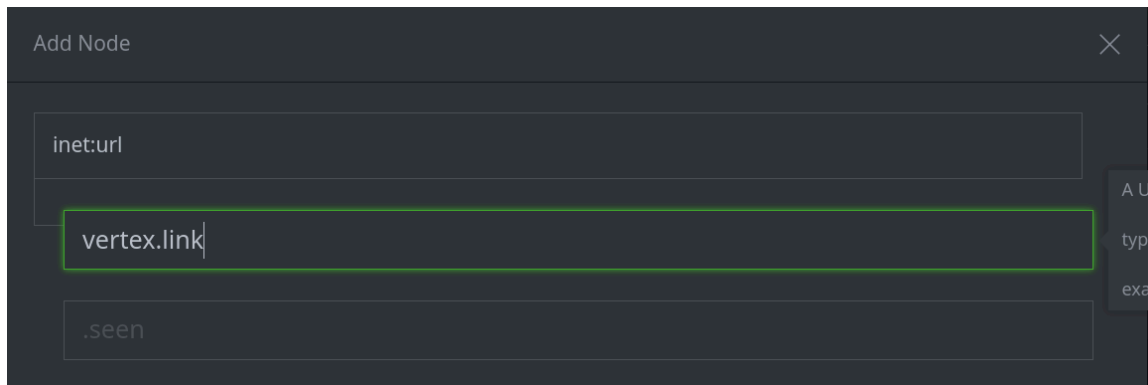
- Choose **inet:url** from the list of forms as the kind of node you want to create:



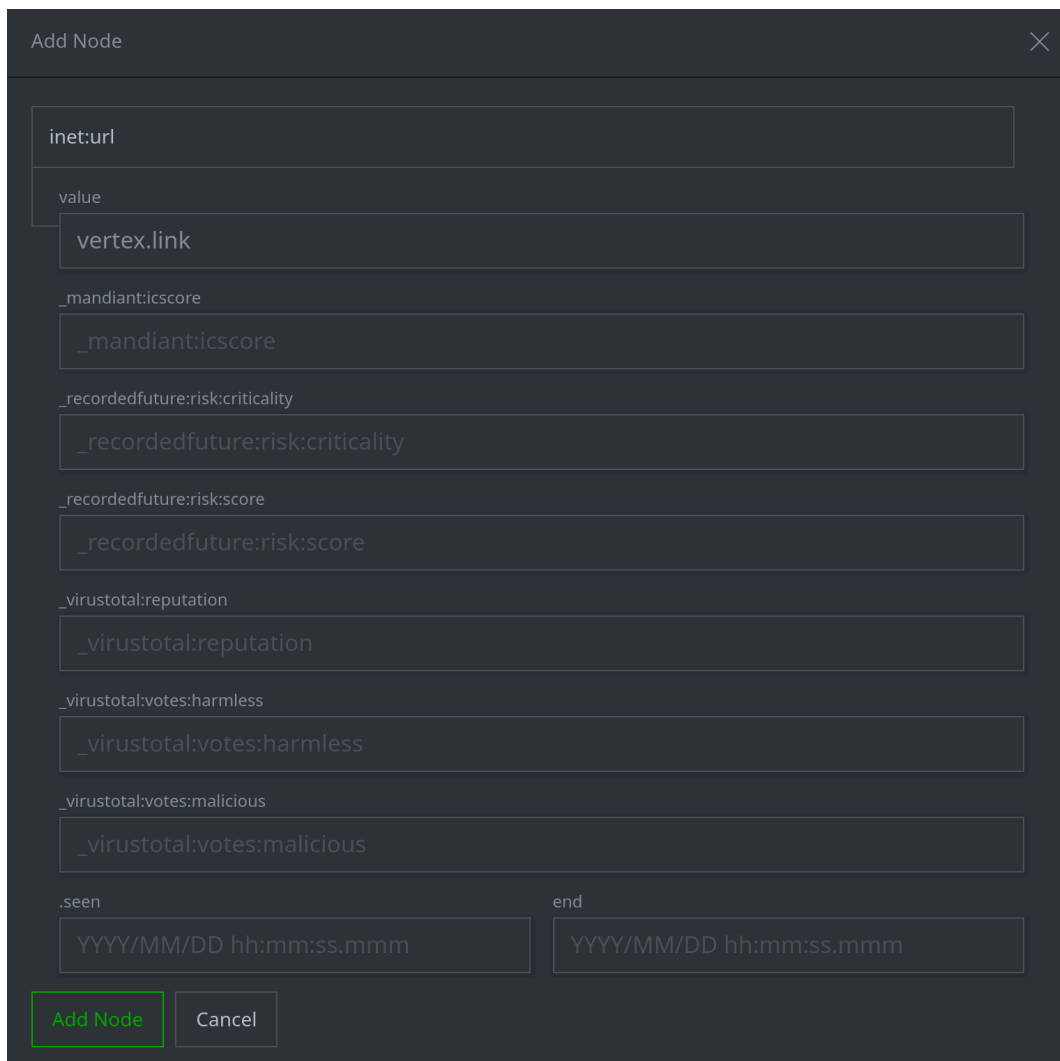
- In the **Add Node** dialog, enter the following in the *value* field:

vertex.link

Your dialog should look similar to the image below:



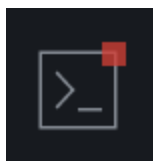
- Click the **Add Node** button to create the node:



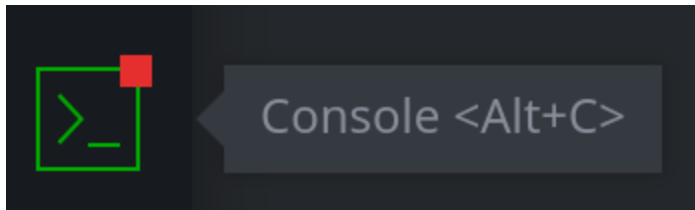
Question 1: What happens when you click the **Add Node** button? Did Synapse create the **inet:url** node?

- Click **Cancel** to close the **Add Node** dialog:

- In your **Toolbar**, the icon for your **Console Tool** has a red square in the upper right corner. This indicates there is an error message present:

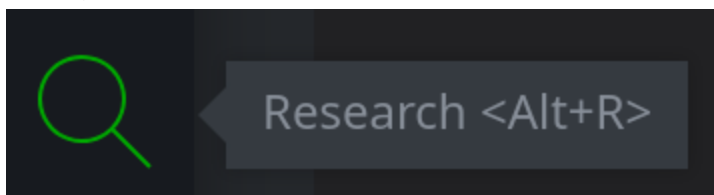


- Click the **Console Tool** icon to open the Tool:

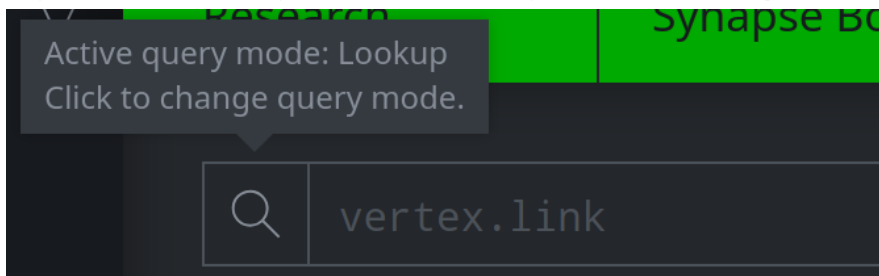


Question 2: What is the error message in the Console Tool? What does it mean?

- From your **Toolbar**, select the **Research Tool**:



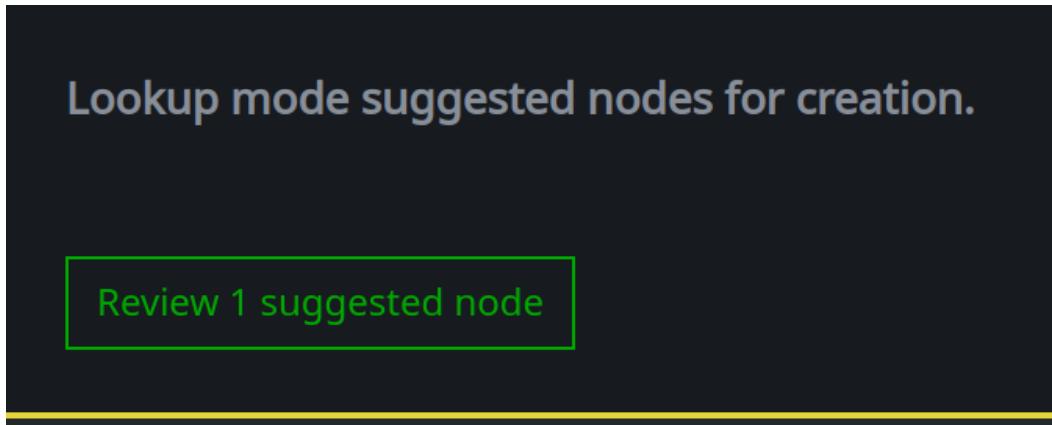
- In your **Storm Query Bar**, ensure that you are in **Lookup** mode:



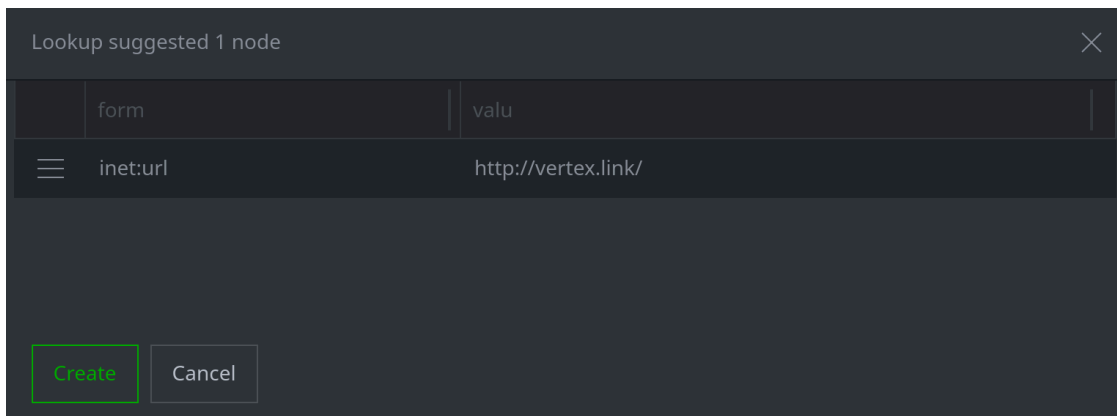
- Enter the following into the **Storm Query Bar** and press **Enter** to run the query:

```
HTTP://VerTEx.LiNK/
```

- In the pop-up ("toast") dialog, click the **Review 1 suggested node** button:



- Click the **Create** button to create the node:

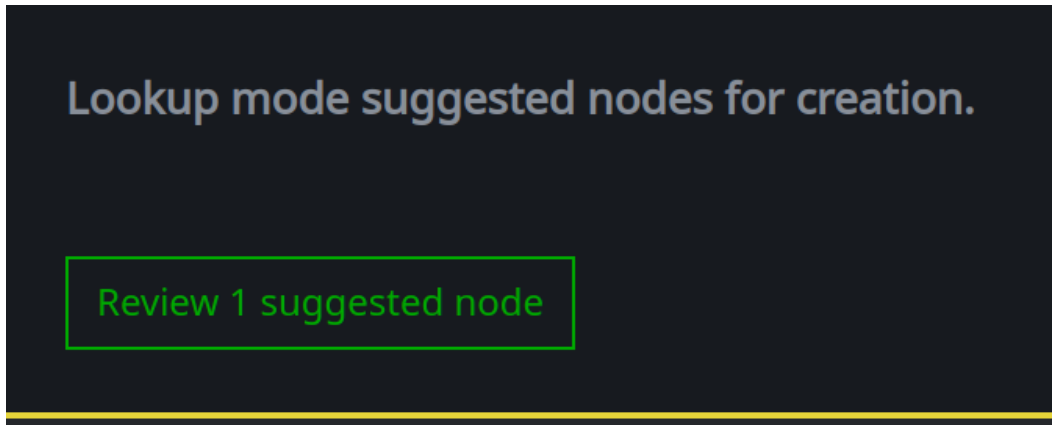


Question 3: Did Synapse create the `inet:url` node? If so, did Synapse modify the data in any way?

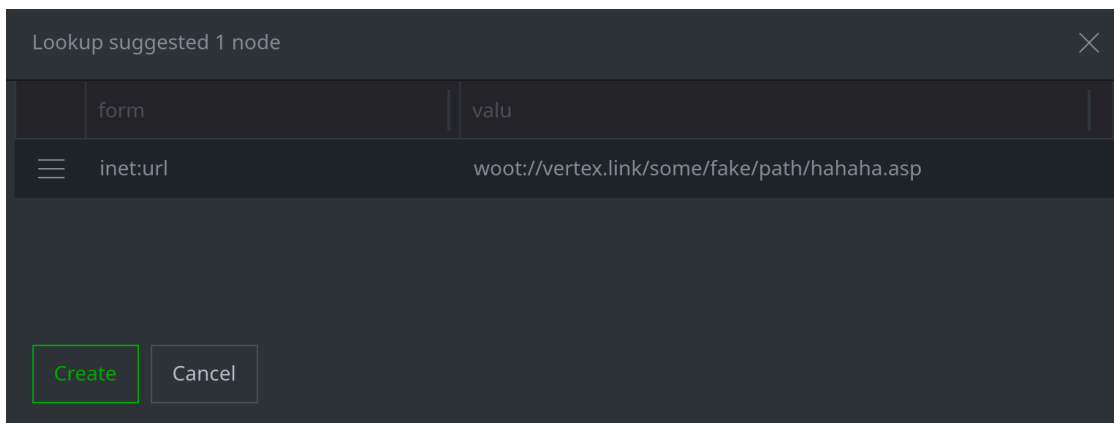
- Enter the following into the **Storm Query Bar** and press **Enter** to run the query:

```
woot://vertex.link/some/fake/path/hahaha.asp
```


- In the pop-up ("toast") dialog, click the **Review 1 suggested node** button:



- Click the **Create** button to create the node:

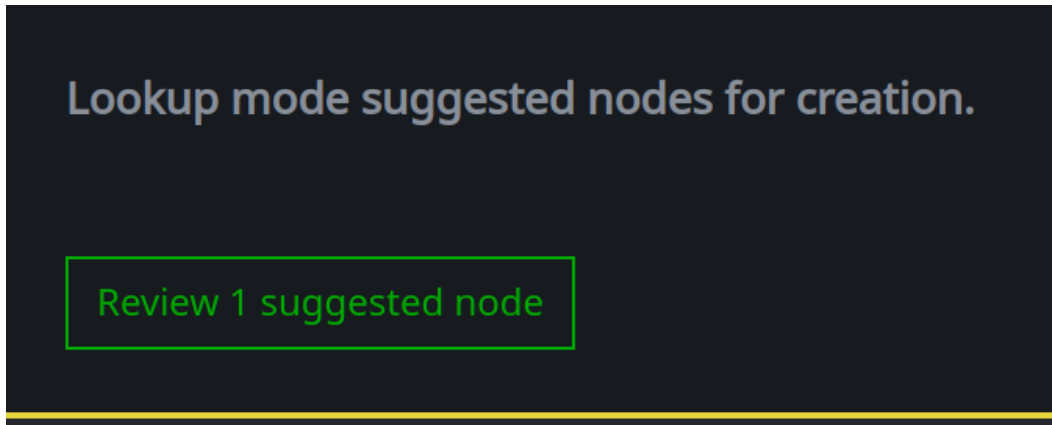


Question 4: Did Synapse create the **inet:url** node? If so, what properties did Synapse set on the node?

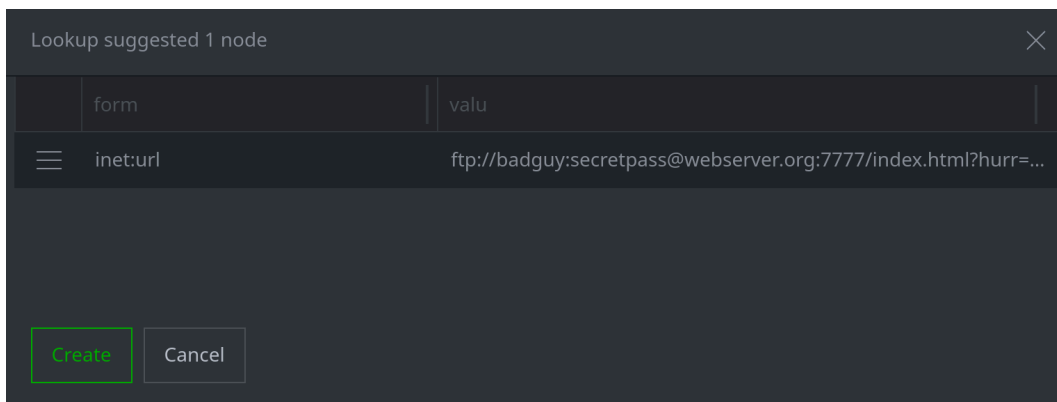
- Using **Lookup** mode, enter the following into the **Storm Query Bar** and press **Enter** to run the query:

```
ftp://badguy:secretpass@webserver.org:7777/index.html?hurr=derp&faz=baz
```

- In the pop-up ("toast") dialog, click the **Review 1 suggested node** button:



- Click the **Create** button to create the node:



Question 5: Did Synapse allow you to create the **inet:url** node? If so, what properties did Synapse set on the node?

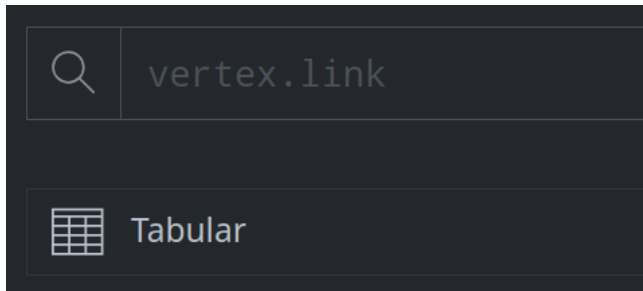
Type-Specific Behavior

Exercise 6

Objective:

- Observe one example of type-specific behavior implemented in Synapse to simplify working with certain kinds of data (in this case, IPv4 addresses).

- In the **Research Tool (Tabular mode)**, ensure your **Storm Query Bar** is in **Lookup** mode:



- With the query bar in **Lookup mode**, run **each** of the following queries separately.

Question 1: in each case, using **Lookup** mode, what (if anything) does Synapse display?

An IP address:

58.247.237.192

An IP address represented in hexadecimal:

0x3AF7EDC0

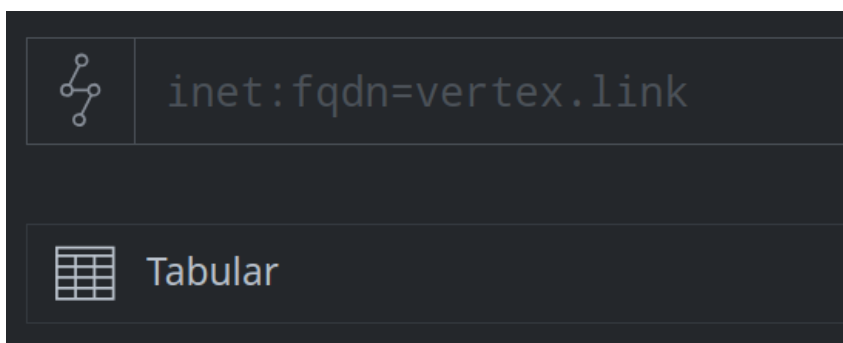
A CIDR address:

58.247.237.0/24

An IPv4 network range:

58.247.237.56-58.247.237.64

-
- In the **Research Tool (Tabular mode)**, ensure your **Storm Query Bar** is in **Storm** mode:



- With the query bar in **Storm mode**, run **each** of the following queries separately.

Question 2: In each case, using **Storm** mode, what (if anything) does Synapse display?

An IP address:

```
inet:ipv4=58.247.237.192
```

An IP address represented in hexadecimal:

```
inet:ipv4=0x3AF7EDC0
```

A CIDR address:

```
inet:ipv4=58.247.237.0/24
```

An IPv4 network range:

```
inet:ipv4=58.247.237.56-58.247.237.64
```

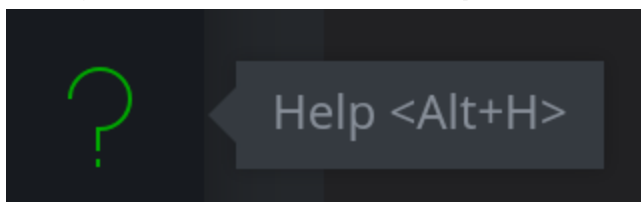
Type Awareness

Exercise 7

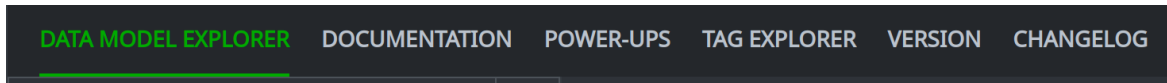
Objective:

- Use the Data Model Explorer to identify nodes that are "connected" by properties that share the same type.
- Understand that these are forms Synapse can easily Explore (or pivot) between by using type awareness.

- From your **Toolbar**, select the **Help Tool**:



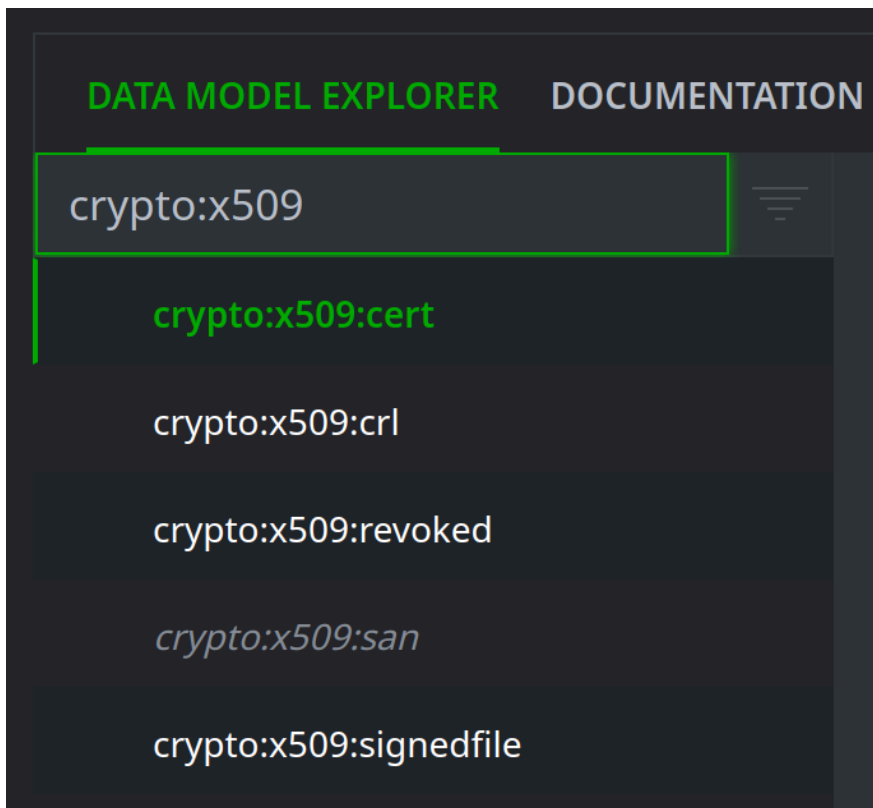
- In the **Help Tool**, click the **Data Model Explorer** tab:



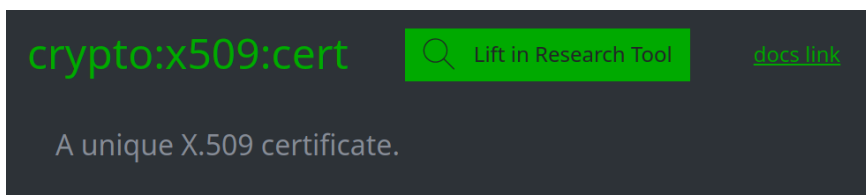
- Enter the following in the *Search* field:

crypto:x509

- Select **crypto:x509:cert** from the list view:



- Review the data model information for this form:



The **crypto:x509:cert** form represents the **metadata** (important x509 certificate details) for an x509 certificate file (the certificate itself is represented by a **file:bytes** node).

- Review the **Properties** section for the **crypto:x509:cert** form:

name	ro	type	doc
:algo		iso:oid	The X.509 si
:crl:urls		(inet:url ,)	The extracte
:ext:crls		(crypto:x509:san ,)	A list of Su
:ext:sans		(crypto:x509:san ,)	The Subject
:file		file:bytes	The file tha
:identities:emails		(inet:email ,)	The fused li
:identities:fqdns		(inet:fqdn ,)	The fused li
:identities:ipv4s		(inet:ipv4 ,)	The fused li
:identities:ipv6s		(inet:ipv6 ,)	The fused li
:identities:urls		(inet:url ,)	The fused li

This section lists all the **properties** for the form (and the associated type).

In many cases, a form's properties are **also** their own forms. Where this is the case, the property **type** is shown in **green** and hyperlinked to the appropriate form in Data Model Explorer.

A **crypto:x509:cert** is **connected** to the forms represented by its properties. This means that you can **navigate** from a **crypto:x509:cert** node to the nodes associated with its properties.

Question 1: Based on the information in **Data Model Explorer**, can you navigate (i.e., using the **Explore** button, or a Storm query) between a **crypto:x509:cert** node and the SHA1 fingerprint (**hash:sha1**) of the certificate?

- In **Data Model Explorer**, for the **crypto:x509:cert** form, review the **Referenced**

By section:

Referenced By		
form	prop	doc
crypto:x509:cert	:issuer:cert	The certificate used by
crypto:x509:revoked	:cert	The certificate revoked
crypto:x509:signedfile	:cert	The certificate for the
inet:flow	:src:ssl:cert	The x509 certificate sen
inet:flow	:dst:ssl:cert	The x509 certificate sen

This section lists all the **forms that have a property** that is a **crypto:x509:cert**. The form names (in **green**) are hyperlinked to their Data Model Explorer entries.

A **crypto:x509:cert** is **connected** to any form that has a **crypto:x509:cert** as a **property**. This means that you can **navigate** from a **crypto:x509:cert** node to any nodes that have the certificate as a property.

Question 2: Can you navigate from a **crypto:x509:cert** node to nodes that show the certificate was used to **sign a particular file**?
