Vertex

Synapse Bootcamp - Module 7

Pre-Storm Background - Answer Key

Pre-Storm Background - Answer Key **1**
Answer Key **2**
Form Categories 2
Exercise 1 Answer 2
Form and Property Namespaces 7
Exercise 2 Answer 7
Exercise 3 Answer 8
Exercise 4 Answer 9
Type Enforcement 10
Exercise 5 Answer 10
Type-Specific Behavior 15
Exercise 6 Answer 15
Exercise 7 Answer 21

© 2025 The Vertex Project, LLC

Answer Key

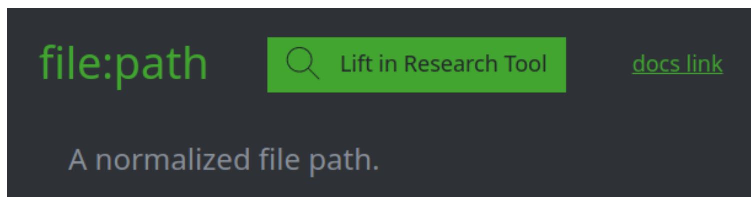
Form Categories

Exercise 1 Answer

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

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

- A **`file:path`** form represents "a normalized file path". That is, the path used to reference a file on a file system:

  A **`file:path`** can be as simple as a file name by itself, or may include directory and / or drive paths.

  All of these can be **`file:path`** nodes:

  ```
  cmd.exe
  /home/bob/Documents/myfile.txt
  C:\Windows\System32\drivers\acpi.sys
  ```

  Normalized means that Synapse **standardizes** the way it stores and represents a file path. In Synapse:
  - all paths are converted to **lowercase,** and
  - all directory separators are converted to **forward slashes:**

    ```
    /home/bob/documents/myfile.txt
    c:/windows/system32/drivers/acpi.sys
    ```

© 2025 The Vertex Project, LLC

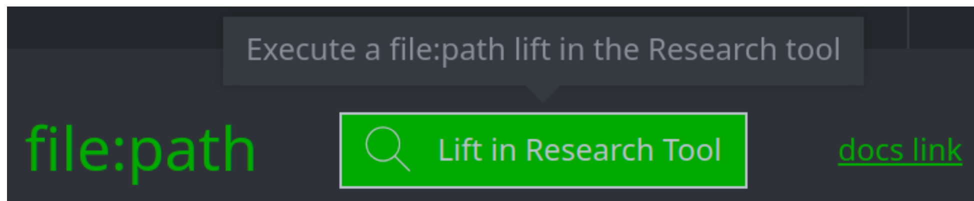
> **Note:** You can **enter** and **query** file paths however you like (upper case, mixed case, forward slashes or backslashes). Synapse recognizes all of these formats and handles the normalization for you!

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

- The primary property value of a **`file:path`** node is the file path. You can see this in the "example" in Data Model Explorer:

```
type: file:path
opts: {"enums":null,"regex":null,"lower":false,"strip":false,"replace":
[],"onespace":false,"globsuffix":false}
example: c:/windows/system32/calc.exe
```

You can also use the **Lift in Research Tool** button to view example **`file:path`** nodes:



> A **`file:path`** form is a "simple" form (i.e., where the primary property value is the "thing" itself).

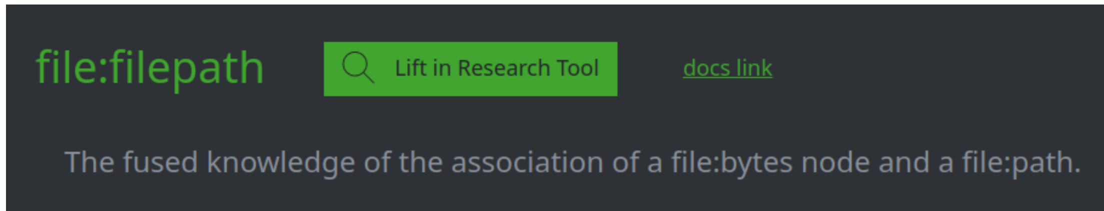
Question 3: Is a **`file:path`** an object, a relationship, or an event?

- A **`file:path`** is an **object.** It represents a file and / or directory path.

> "Simple" forms often represent objects.

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

- A **`file:filepath`** form represents "the fused knowledge of the association of a **`file:bytes`** node and a **`file:path`**":

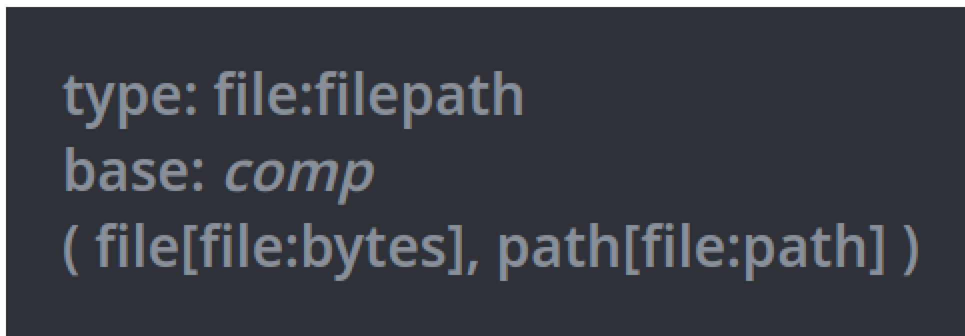


In simple terms: a **`file:filepath`** node links a **file** (**`file:bytes`**) to a **path** (**`file:path`**) where the file was seen. This can represent:

- A file (set of bytes) that was observed with a specific file name (e.g., `svchost.exe` or `20240412_invoice.pdf`); or
- A file (set of bytes) that was observed in a specific location (e.g., `c:\windows\temp\download.exe` or `/home/jsmith/Documents/myfile`).

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

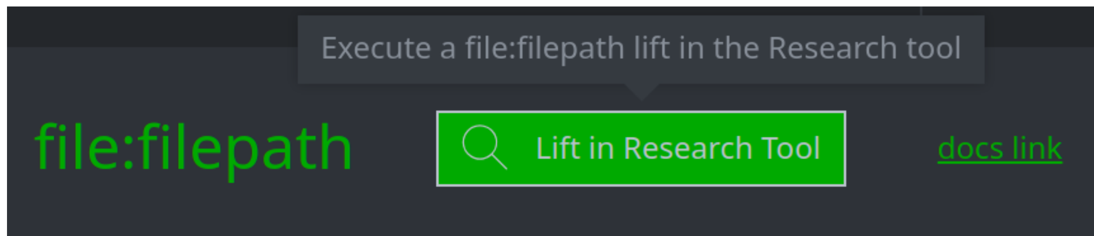
- The primary property value of a **`file:filepath`** node is the combination of the **file** (**`file:bytes`**) and its **path** (**`file:path`**):



This is represented by the example primary property format shown above:
`( file[file:bytes], path[file:path] )`

© 2025 The Vertex Project, LLC

You can view example **`file:filepath`** nodes using the **Lift in Research Tool** button:



A **`file:filepath`** is a **comp** (composite) form. Composite forms have primary properties made up of more than one value (usually two).

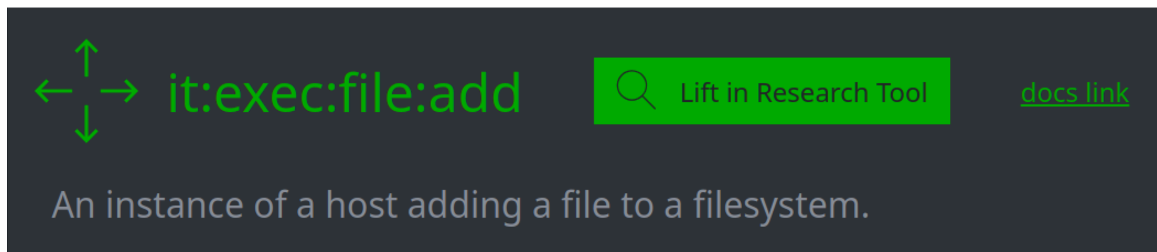
Question 6: Is a **`file:filepath`** an object, a relationship, or an event?

- A **`file:filepath`** is a **relationship** between a file and its observed file system path or file name.

Composite forms often represent relationships. The primary property consists of the objects (usually two) that share the relationship.

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

- An **`it:exec:file:add`** form represents "an instance of a host adding a file to a filesystem":



In simple terms: it represents **activity** that occurred on a host at a **specific time** ("instance"). The activity was a file added to the file system - for example, due to the execution ("exec") of a program.

A "file add" may occur during activity such as:

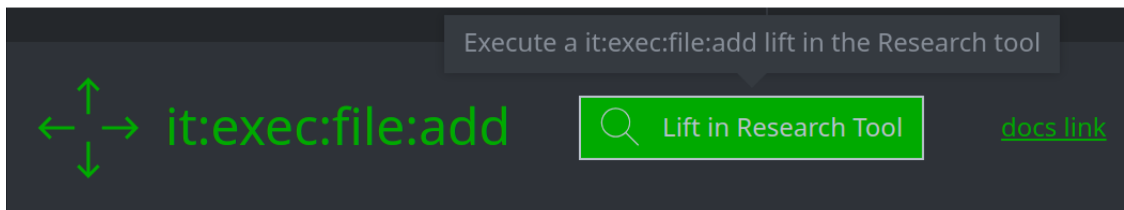
- Installation of new software
- Creating a new document
- Downloading and saving a file

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

- The primary property value of an **`it:exec:file:add`** node is a globally unique identifier, or **guid**:

type: it:exec:file:add
base: *guid*

You can view example **`it:exec:file:add`** nodes using the **Lift in Research Tool** button:



An **`it:exec:file:add`** form is a **guid** form, where the primary property is a 128-bit value (represented in hexadecimal - it looks like an MD5 hash, although it is not an MD5).

Guid forms are used when no single value (or reasonable set of values) are guaranteed to make the form unique.

A guid (128-bit value) is large enough to ensure that each guid form is unique within a Cortex. We can record as much (or as little) information about the node (by setting properties) as we choose.

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

© 2025 The Vertex Project, LLC

- An **`it:exec:file:add`** is an **event.** It represents a specific action (i.e., adding a file to a host filesystem) that occurred at a **specific point in time.**

> In Synapse, forms that represent **events** will often have a **`:time`** property to show that the node represents a "point in time" observation or occurrence.
>
> **Guid** forms are often used to represent events or other "instance" data.

Form and Property Namespaces

Exercise 2 Answer

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.**

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

- The **`inet:`** (Internet) portion of the Synapse data model includes forms related to networking, including:

Kind of Data	Example Synapse Forms

Network addresses
 	
Network address ranges
	`inet:ipv6`
	`inet:mac`
	`inet:cidr4`
	`inet:asnet6`

Clients and servers	`inet:client` `inet:server`

Server infrastructure and properties	`inet:tls:servercert`
	`inet:tls:clientcert`
	`inet:ssl:cert`
	`inet:banner`

Network communications and actions	`inet:flow` `inet:download`

Kind of Data	Example Synapse Forms
Network registration (whois) data	**inet:whois:rec** **inet:whois:iprec**
Network protocols	**inet:dns:*** **inet:http:***
Network-based services and related activity	**inet:service:account** **inet:service:channel** **inet:service:group** **inet:service:message** **inet:service:platform**
Network-based accounts and related activity	**inet:web:acct** **inet:web:group** **inet:web:post**

Note: the **inet:service:*** forms were introduced in June 2024 to update (and eventually replace) the **inet:web:*** forms. You may see either or both kinds of nodes in Synapse during the transition from the older model elements to the current ones.

Similarly, **inet:tls:servercert** and **inet:tls:clientcert** were introduced in April 2024 to update (and eventually replace) the **inet:ssl:cert** form.

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

- There are several "sub-categories" in the **inet:*** category. These include:
    - Specific protocols (**inet:dns:***, **inet:http:***)
    - Email messages / communications (**inet:email:***),
    - Registration data (**inet:whois:***)

...to name a few.

Exercise 3 Answer

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

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

- The **full** property name is:

```
inet:whois:rec:registrant
```

The full property name consists of the **form** (`inet:whois:rec`) and **property** (`:registrant`) names together.

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

- The **relative** property name is:

```
:registrant
```

The relative property name consists of **only** the property name (and its leading character - either colon (`:`), dot (`.`), or colon underscore (`:_`)) **relative to** the form name.

Exercise 4 Answer

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

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

- The **full** property name is:

```
file:bytes:mime:pe:imphash
```

The full property name consists of the **form** (`file:bytes`) and **property** (`:mime:pe:imphash`) names together.

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

- The **relative** property name is:

```
:mime:pe:imphash
```

The relative property name consists of **just** the property name (and its leading character - either colon (`:`), dot (`.`), or colon underscore (`:_`)) **relative to** the form name.

In this case, the property name for the PE import hash is nested within a few "subcategories" that cluster related properties on a file (`file:bytes`) node.

The `:mime` "subcategory" holds properties related to file MIME types. The `:pe` "subcategory" holds properties specific to PE (portable executable) files.

The relative property name must include these "subcategories" - everything **except** the form name (`file:bytes`). You cannot simply use `:imphash`.

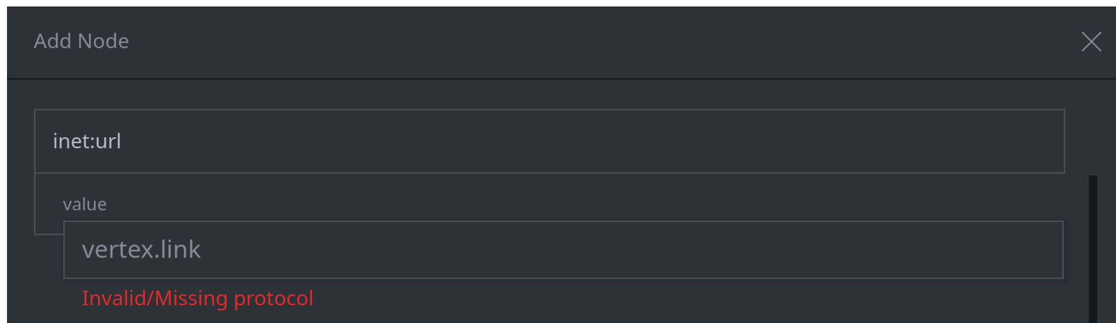
Type Enforcement

Exercise 5 Answer

Objective:
- **Observe how Synapse helps to ensure data is consistent and correct through normalization and type enforcement.**

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

- Synapse gives you an **Invalid/missing protocol** error and does not create the node:



> Before creating any node, Synapse checks its **type enforcement** rules. One rule for URLs is that they must have a protocol header (such as `http://` or `file://`).

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

- The **Console Tool** displays the full error message:

```
ERROR: BadTypeValu: Invalid/Missing protocol
```

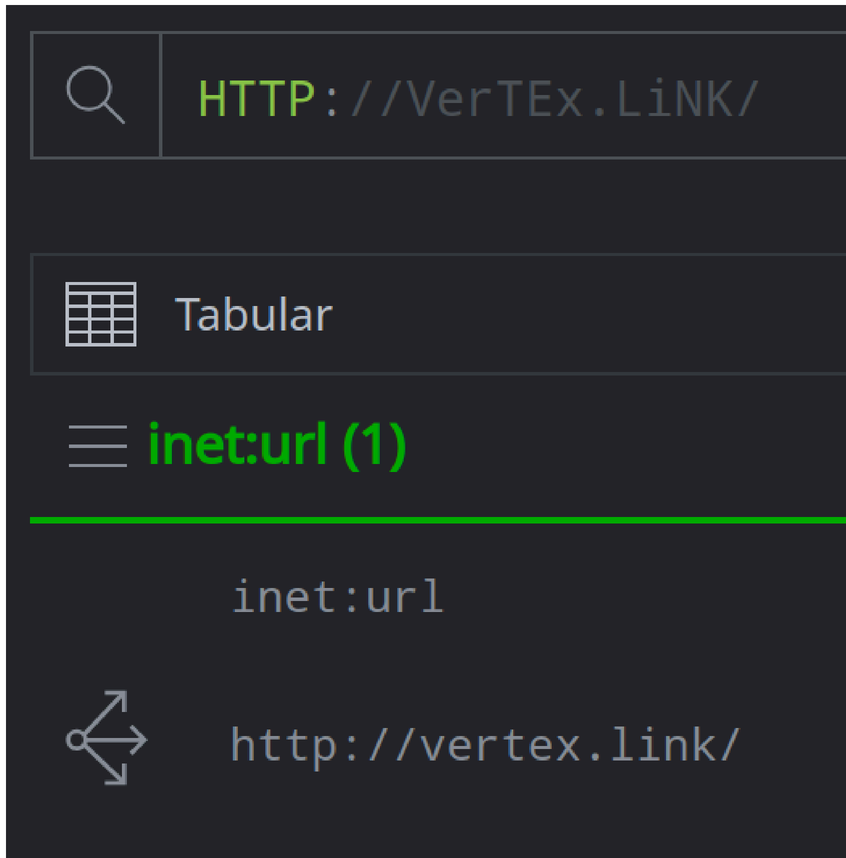
`BadTypeValu` refers to Synapse's **type enforcement** - the rules Synapse follows to make sure that data of a particular **type** (in this case, `inet:url`) is "well formed" for the kind of data you're entering.

In this case the **value** you entered (`vertex.link`) is **bad** (invalid) for the **type** of node you're creating (an `inet:url`).

Synapse also provides a hint as to what is wrong with your data - in this case, there is no protocol header (such as **ftp://**).

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

- Synapse created the **`inet:url`** node, but changed the URL to lowercase:



Synapse **normalizes** the URL by converting it to lowercase. By converting to lowercase, Synapse ensures that you don't end up with more than one node for the same URL where the only difference is the case used.

Normalizing data is **not** the same as type enforcement. Converting to lowercase for consistency has nothing to do with making sure a URL looks like a URL.

Normalizing data makes Synapse life simpler for you because URLs are always stored the same way - in lowercase. If you are writing a Storm query or searching for a node, you do not have to worry about matching the exact case of the URL!

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

© 2025 The Vertex Project, LLC

- Synapse created the node. The **`inet:url`** node has the following properties:

NODE ALL TAGS ALL PROPS

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

:base      woot://vertex.link/some/fake/path/hahaha.asp
:fqdn      vertex.link
:params
:path      /some/fake/path/hahaha.asp
:proto     woot
.created   2023/11/20 22:49:01.187
```

Even though this is not a "real" URL, it passes Synapse's **type enforcement** checks:

- It has a protocol string - **`woot://`**
- It has a valid host (FQDN / IPv4 / IPv6) - **`vertex.link`**
- It has a valid URL path - **`/some/fake/path/hahaha.asp`**

Type enforcement tries to be "restrictive enough" to catch common errors, but not so restrictive that it prevents you from creating valid nodes. (This is one reason **Lookup mode** will **prompt** you before creating nodes - it gives you a chance to review and **remove** any "bad" nodes before creating anything.)

Type enforcement will not catch **all** bad data. But it **will** catch many inconsistencies ("hey you forgot the FQDN in your URL") or copy/paste errors ("looks like you tried to make a URL out of an email address by mistake").

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

- Synapse created the **`inet:url`** node and set the following properties:

```
NODE   ALL TAGS   ALL PROPS

▪ inet:url
  "ftp://badguy:secretpass@webserver.org:7777/index.html?
  hurr=derp&?faz=baz"

▪ :base       ftp://badguy:secretpass@webserver.org:7777/index.html
▪ :fqdn       webserver.org
▪ :params     ?hurr=derp&?faz=baz
▪ :passwd     secretpass
▪ :path       /index.html
▪ :port       7777
▪ :proto      ftp
▪ :user       badguy
▪ .created    2023/11/20 22:53:07.320
```

Synapse is able to parse out the different parts of the URL into various secondary properties. In this URL, Synapse recognizes (and models!) the protocol, credentials (username and password), FQDN / hostname, port number, file path, and parameters used in the query. Yay Synapse!

This is the positive side of **type enforcement**. Type enforcement can help **prevent** you from creating nodes with bad or malformed data. In addition, when data **is** well-formed, Synapse can **automatically** extract and represent lots of useful things for you!

© 2025 The Vertex Project, LLC

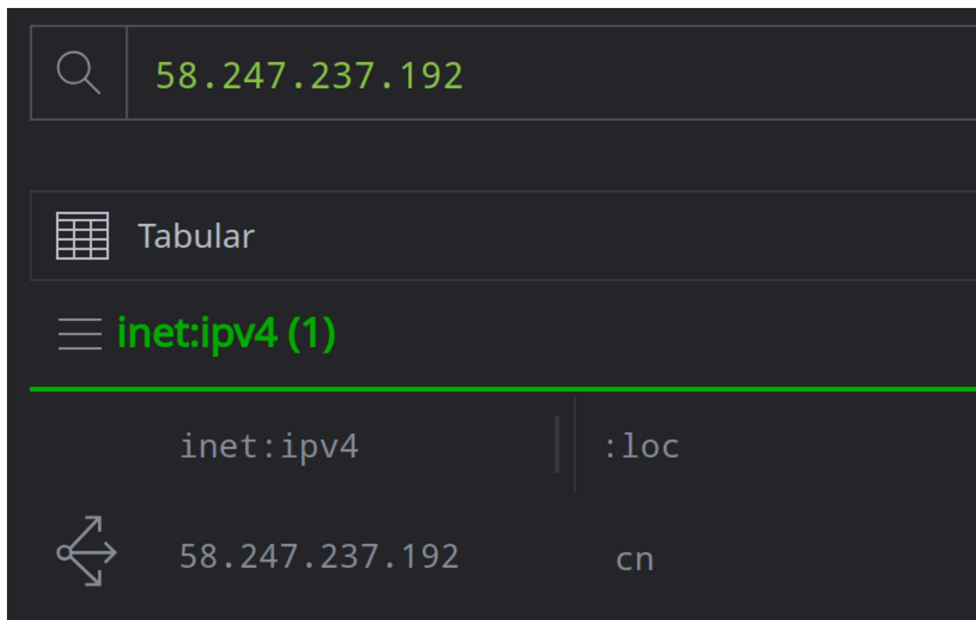
Type-Specific Behavior

Exercise 6 Answer

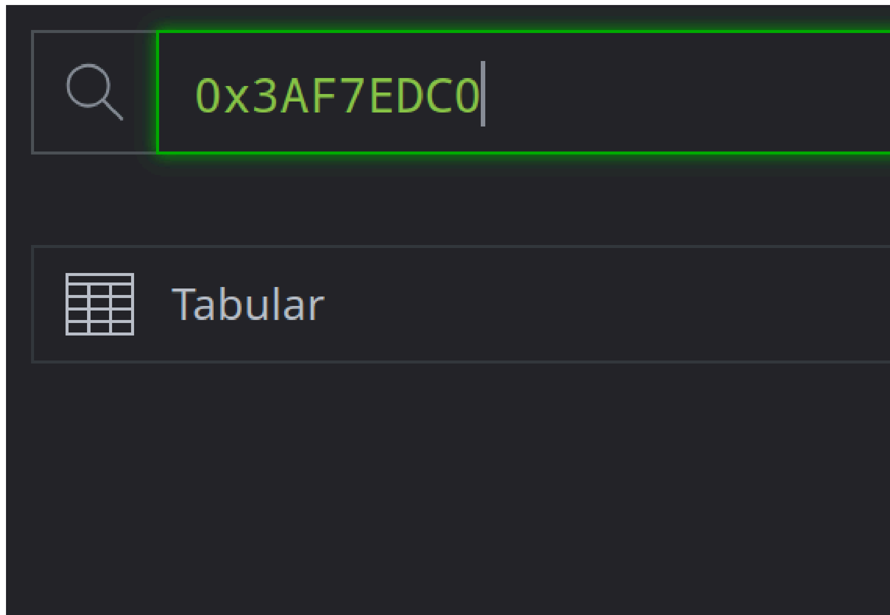
Objective:
- **Observe one example of type-specific behavior implemented in Synapse to simplify working with certain kinds of data (in this case, IPv4 addresses)**

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

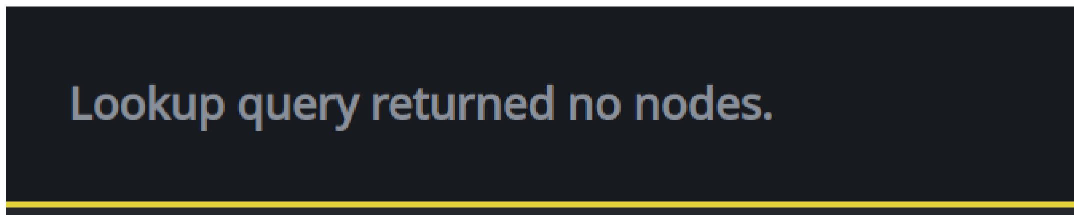
- For **58.247.237.192**, Synapse displays the node for this IPv4 address:



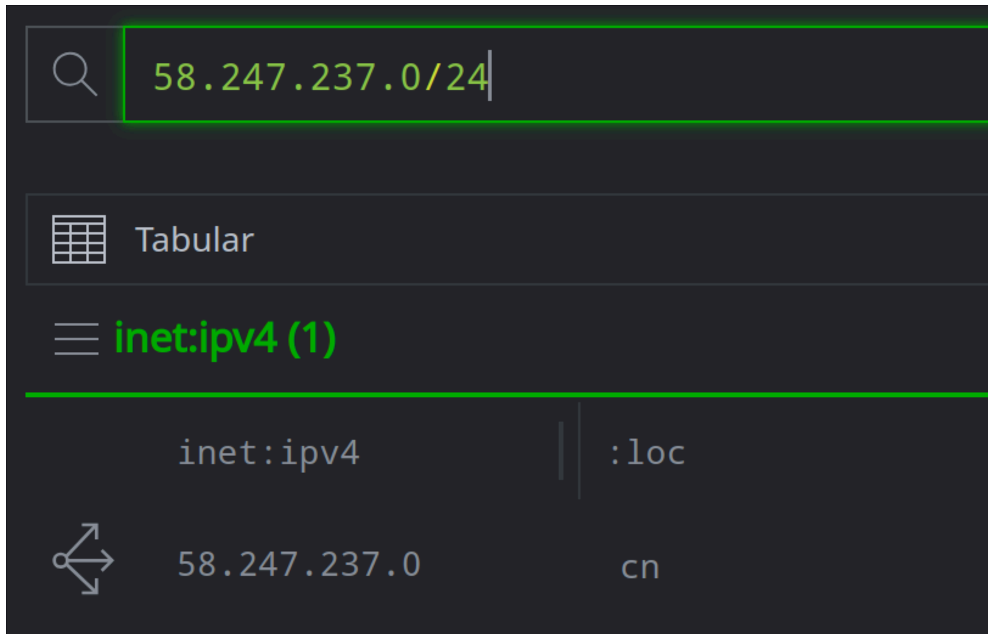
- For `0x3AF7EDC0`, Synapse does not display anything:



A pop-up ("toast") message confirms that no nodes were returned:



- For **`58.247.237.0/24`**, Synapse displays the single IPv4 **`58.247.237.0`**:



- For **`58.247.237.56-58.247.237.64`**, Synapse displays the IPv4 **`58.247.237.56`**:

© 2025 The Vertex Project, LLC

In **Lookup** mode, Synapse can only recognize individual IPv4 addresses in dotted-decimal format. Lookup mode does not understand other formats (like hexadecimal) and cannot recognize things like CIDR format or ranges.

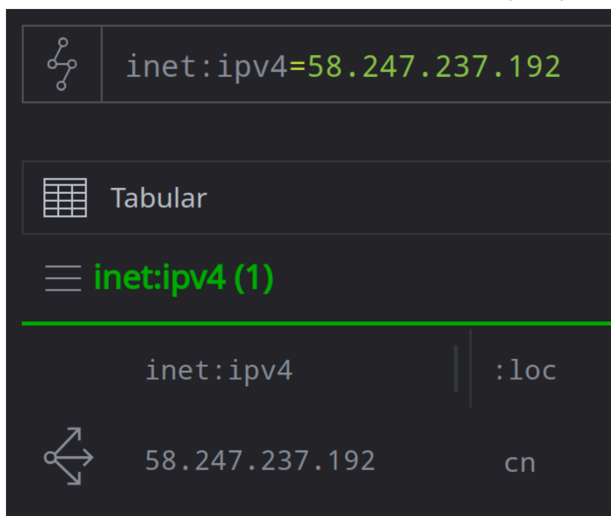
Lookup mode uses logic (regular expressions) to identify common values such as IPv4s or MD5 hashes. In other words, Synapse makes a best guess at "what you mean" based on the data you enter. In Lookup mode, you do not **tell** Synapse that the data you paste in is an IPv4 address - Synapse has to try to figure that out.

The forms Synapse can recognize must be narrowly defined. In other words, Synapse can only go so far in trying to "guess what you mean" when you enter values in these modes.

For example, in the example above the hexadecimal string represents an IPv4 address. In other circumstances it could be a string or a memory address.

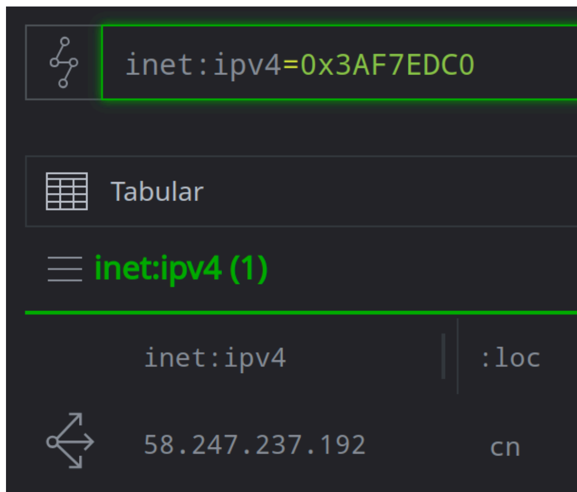
Question 2: In each case, using **Storm** mode, what (if anything) does Synapse display?

- For **`inet:ipv4=58.247.237.192`**, Synapse displays the single IPv4 address:

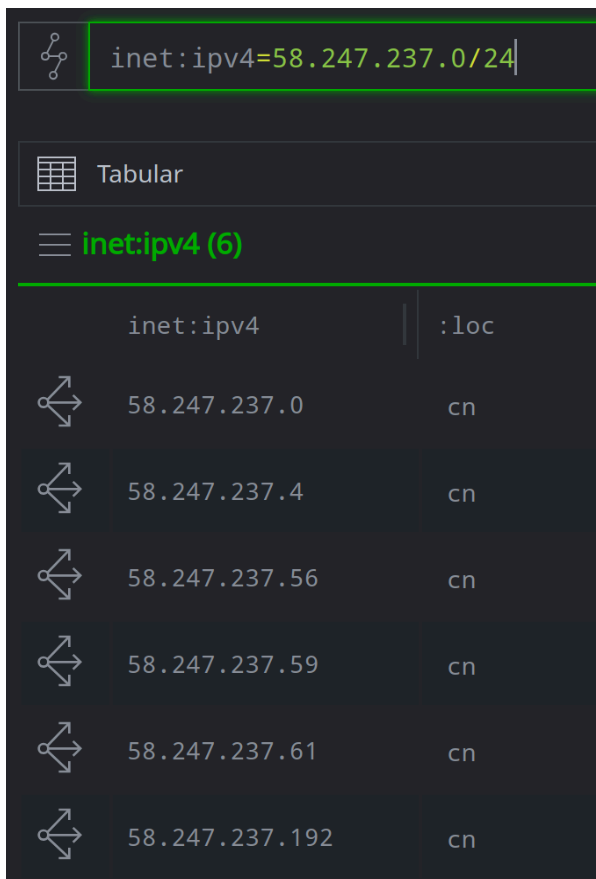


© 2025 The Vertex Project, LLC

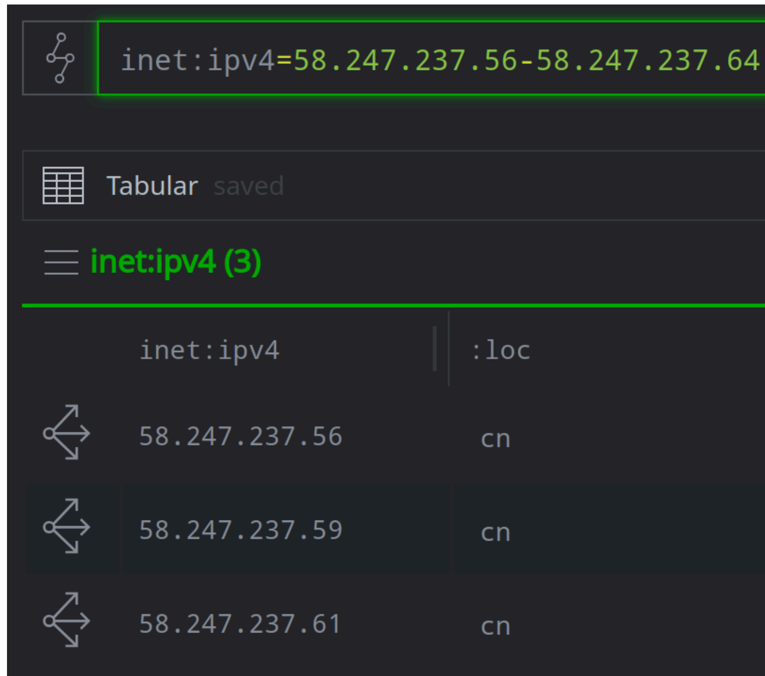
- For **`inet:ipv4=0x3AF7EDC0`**, Synapse recognizes the hexadecimal format and shows you the associated IPv4:



- For **`inet:ipv4=58.247.237.0/24`**, Synapse recognizes CIDR notation as a valid way to specify a **range** of IPv4 addresses. Synapse shows you **all IPv4s in that range that exist** within Synapse:



- For **`inet:ipv4=58.247.237.56-58.247.237.64`**, Synapse recognizes this as another way to specify a **range** of IPv4s, and shows you **all IPv4s in that range that exist** within Synapse:



In **Storm mode,** you include the **form** (**`inet:ipv4`**) as part of your query.

By using the form, you **"tell"** Synapse what kind of data you are asking about. Synapse "knows" you are entering IPv4 addresses; so Synapse can use **type-specific behavior** to do things like recognize alternate formats for IPv4s (such as hexadecimal or CIDR notation).

One way that **type-specific behavior** makes things easier is by allowing you to **input** data in a variety of formats. Synapse understands different ways to represent data. You can enter the data as it appears - you do not need to manually convert it.

In the case of **`inet:ipv4`** addresses, Synapse's ability to recognize ranges and CIDR notation allows you to easily specify **sets** of IPv4 addresses without having to list them individually.

A lot of Synapse's **type-specific behavior** is designed for very specific (but helpful) use cases. You do not need to memorize these use cases, but you should know where to find them when you need them! They are described in the Storm Reference of the Synapse User Guide under Type-Specific Behavior.

Exercise 7 Answer

Objective:

- **Use the Data Model Explorer to identify nodes that are "connected" by properties that share the same type (i.e., forms that Synapse can readily Explore or pivot between by using type awareness).**

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?

- **Yes.**

  The **`crypto:x509:cert`** node's **`:sha1`** property represents its SHA1 fingerprint. You can navigate from the certificate (**`crypto:x509:cert`**) to the **`hash:sha1`** node

of its fingerprint using the **:sha1** property:

Properties

name	ro	type
:algo		iso:oid
:crl:urls		(inet:url,)
:ext:crls		(*crypto:x509:san*,)
:ext:sans		(*crypto:x509:san*,)
:file		file:bytes
:identities:emails		(inet:email,)
:identities:fqdns		(inet:fqdn,)
:identities:ipv4s		(inet:ipv4,)
:identities:ipv6s		(inet:ipv6,)
:identities:urls		(inet:url,)
:issuer		str
:issuer:cert		crypto:x509:cert
:md5		hash:md5
:rsa:key		rsa:key
:selfsigned		bool
:serial		hex
:sha1		hash:sha1
:sha256		hash:sha256
:signature		hex

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

- **Yes.**

  A **crypto:x509:signedfile** node is a "relationship" node that links a file (**file:bytes** node) to the certificate used to sign the file. You can navigate from the certificate (**crypto:x509:cert**) to any **crypto:x509:signedfile** nodes

using their `:cert` property:

Referenced By

form	prop
crypto:x509:cert	:issuer:cert
crypto:x509:revoked	:cert
crypto:x509:signedfile	:cert
inet:flow	:src:ssl:cert
inet:flow	:dst:ssl:cert