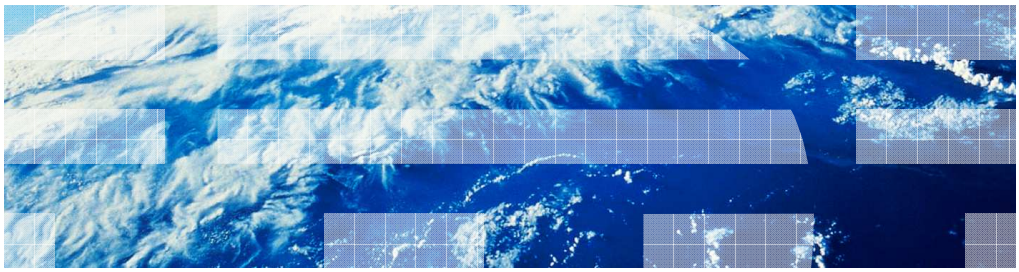
IBM Tivoli Network Manager IP Edition V3.8

Enhancing discovery with pattern matching



Tivoli software

Welcome to the IBM Education Assistant module for Tivoli Network Monitoring IP Edition Version 3.9. This module is about using pattern matching to enhance discovery data.

After you complete this module, you should be able to:

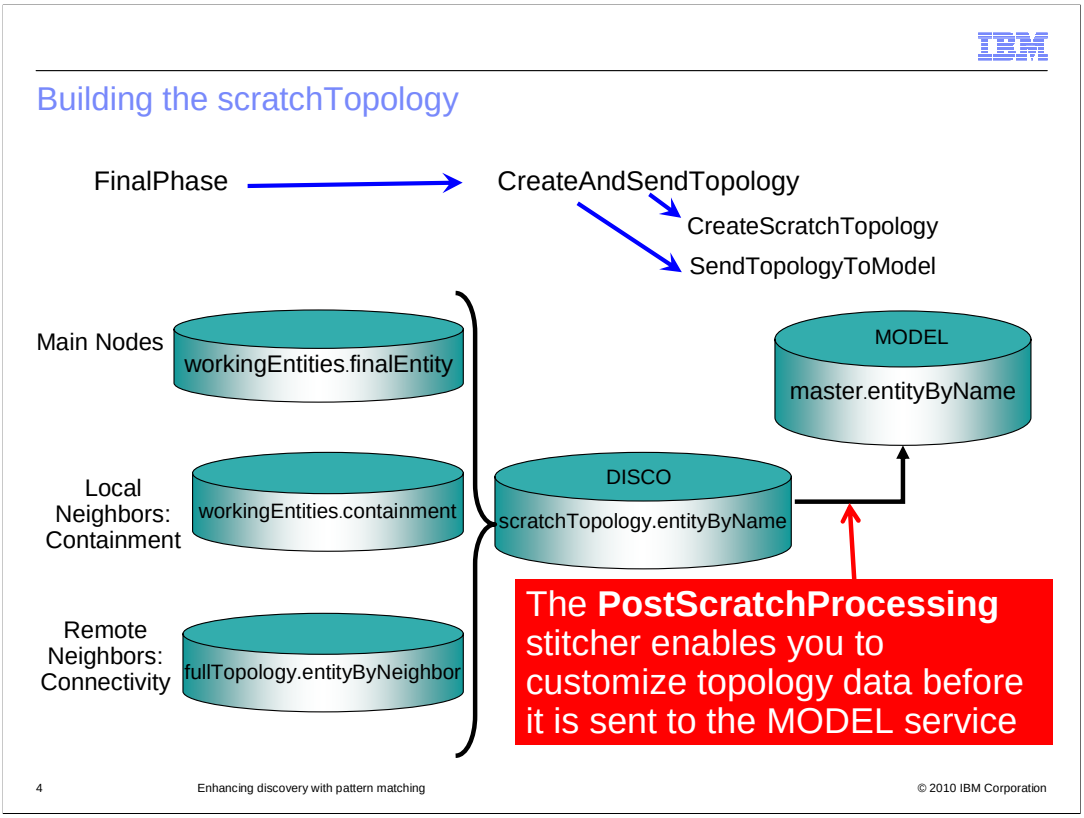
Identify the file that is used to run custom discovery stitchers

Use a simple discovery stitcher to add information to a network discovery using pattern matching

Section

Customizing discovery information

Many IBM Tivoli® Network Manager users want to add business information to the network entities discovered by the software. This module demonstrates one example of how to customize discovery information.



In general, there are three principal steps for performing a network discovery.

1.The finders find devices on the network.

2.The agents interrogate devices on the network to determine the types of entities and the connectivity between them. Each chassis, interface, or logical interface is an entity in the discovery database.

3.The stitchers populate databases with information about the chassis and interface entities. They also populate information about the things contained within entities (such as a card inside of a chassis), and the connectivity between entities.

A **FinalPhase** stitcher runs to build the final topology. The discovery service (DISCO) creates a scratch topology. This topology is sent to the MODEL service, and the master.entityByName table serves as the complete topology record.

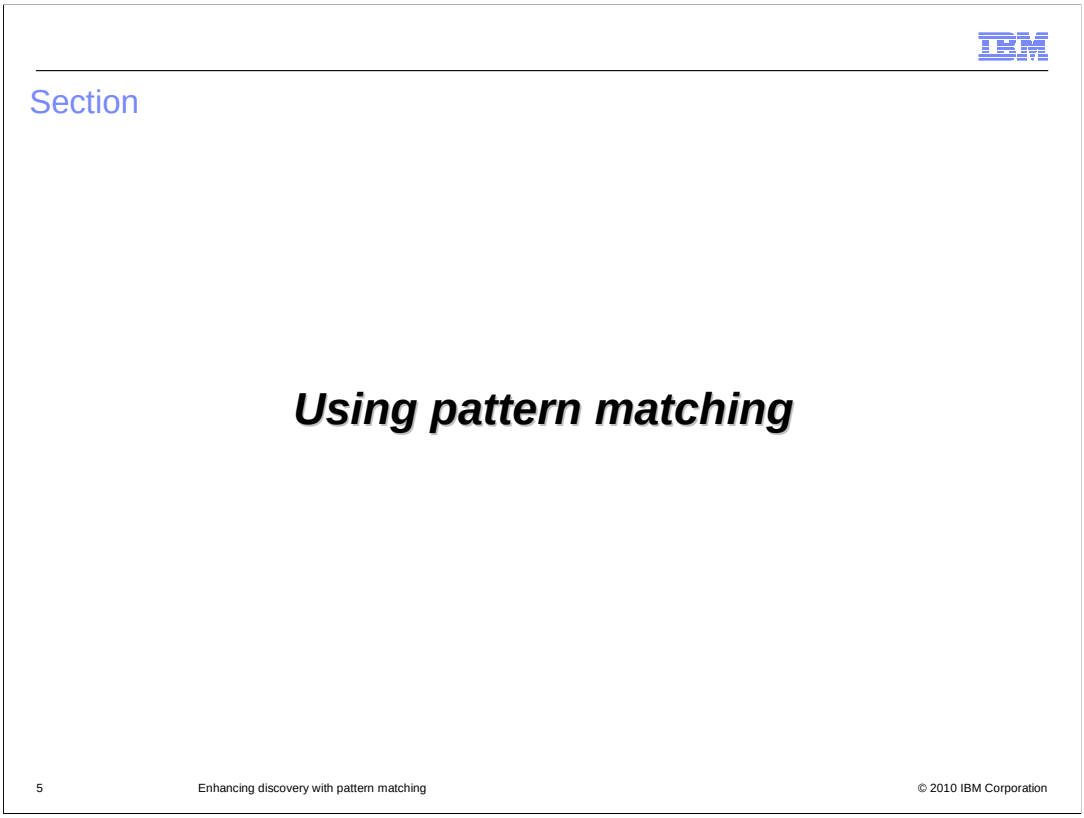
The **CreateAndSendTopology stitcher** is a final phase stitcher that has three main steps:

1.It builds the entities.

2.It builds the layers or connections between the entities.

3.It calls other stitchers to build a scratch topology and then send that topology to the MODEL service.

If you want to add custom discovery data, you can modify the **PostScratchProcessing** stitcher. This stitcher can modify the scratch topology before it is sent to the MODEL service. You can create custom modular stitchers to perform specific topology enrichment tasks and call these stitchers from the **PostScratchProcessing** stitcher.

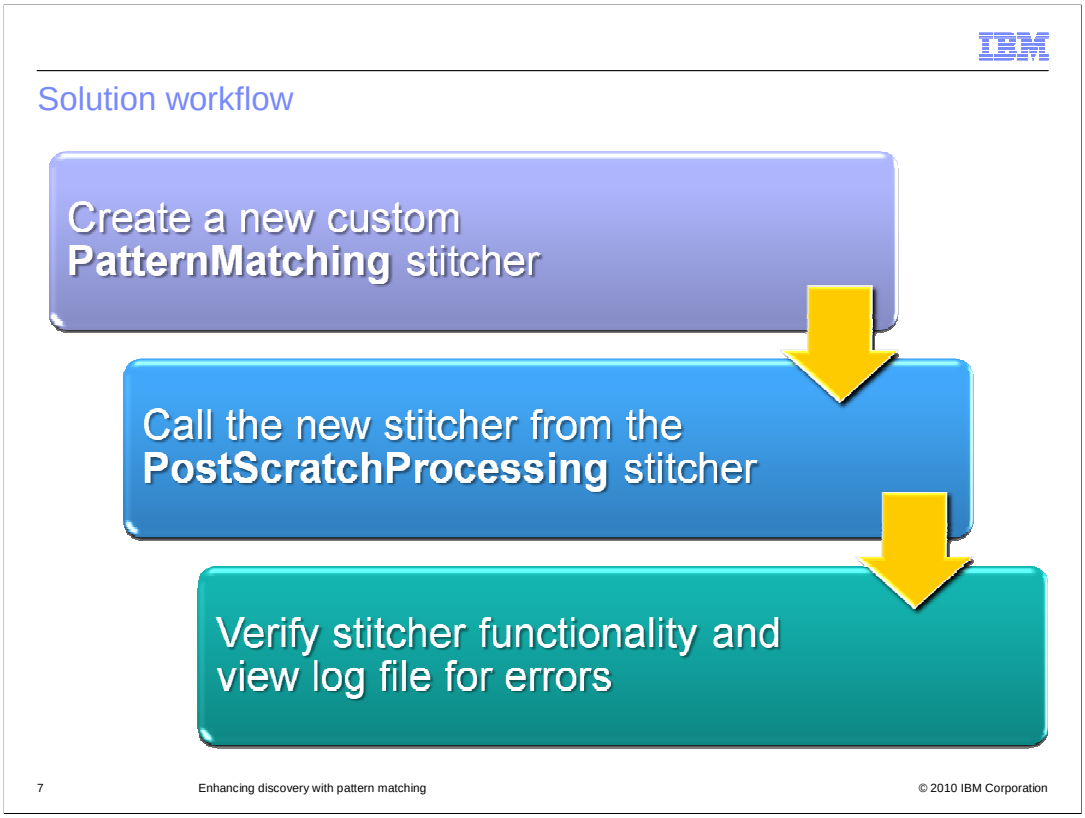


One example of enriching discovery data using a custom stitcher is to derive information using pattern matching on some field in the discovery data.

Pattern matching

- A customer is using a consistent naming schema:
  <DeviceType>-<continent >-<country>-<city>-<numeric>
- Examples:
  - **`r-eu-uk-lon-001`** – Router, Europe, United Kingdom, London
  - **`s-na-us-nyc-002`** – Switch, North America, USA, New York City
  - **`f-eu-uk-lon-001`** – Firewall, Europe, United Kingdom, London
  - **`r-as-ch-bej-003`** – Router, Asia, China, Beijing

Consider the example of a customer using a consistent naming schema. In the example shown here, a customer has a naming schema that reveals characteristics about a device. This naming schema includes the device type, continent, country, and city in which the device is located. The device name also ends with a numeric field to differentiate it from the same type of devices at the same location. In this case, you can build a custom stitcher to extract information from the naming schema.



When making multiple types of enrichments to discovery data, it is best to use a modular design. You can put each discovery enrichment methodology into a separate custom stitcher. This new custom stitcher can be called from the PostScratchProcessing stitcher. After running the new custom stitcher from the PostScratchProcessing stitcher, you can view the discovery log file in the **$NCHOME/log/precision** directory for errors.

Begin by creating a custom stitcher. Name the stitcher to indicate its basic function. For this example, the stitcher is named **HostPatternMatching** because it will apply pattern matching to discovered host names.

Initializing variables

- When declaring integer variables, initialize with zero
- When declaring string variables, initialize with NULL

```
int count=0;
text host = NULL;
text domainPattern = NULL;
```


***In the example of a customer using a consistent naming schema, you can begin by creating a HostPatternMatching stitcher. In any stitcher that you write, always initialize integer variables with a value of zero. Initialize all text variables with the value of NULL. Topology string variables that will be populated by the stitcher should have a default value set to “unknown.” This way, if a value for an entity cannot be determined, the value will be shown as “unknown.”

Setting default values to UNKNOWN

```
ExecuteOQL("UPDATE scratchTopology.entityByName
       SET ExtraInfo->m_DeviceType = 'UNKNOWN',
       ExtraInfo->m_Continent = 'UNKNOWN',
       ExtraInfo->m_Country = 'UNKNOWN',
       ExtraInfo->m_City = 'UNKNOWN'
       WHERE EntityType=1 OR EntityType=8;");
```

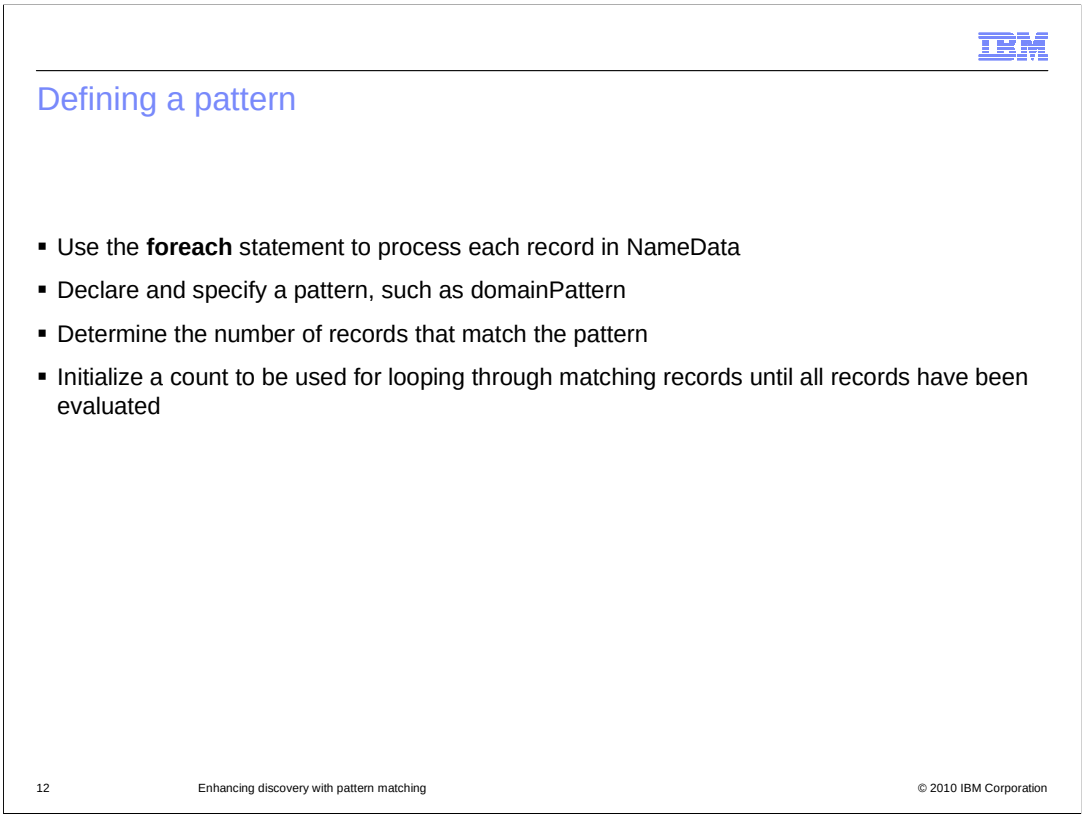

***In this example, the stitcher is used to populate information about device type, continent, country, and city. This information will be used to create custom network partition map views. Because the network map only shows chassis entities, only records for chassis entities need to be modified by the stitcher. An entity type of 1 or 8 refers to a chassis entity. Only entities with these values will be modified by the custom Host**PatternMatching** stitcher.

Populating a RecordList

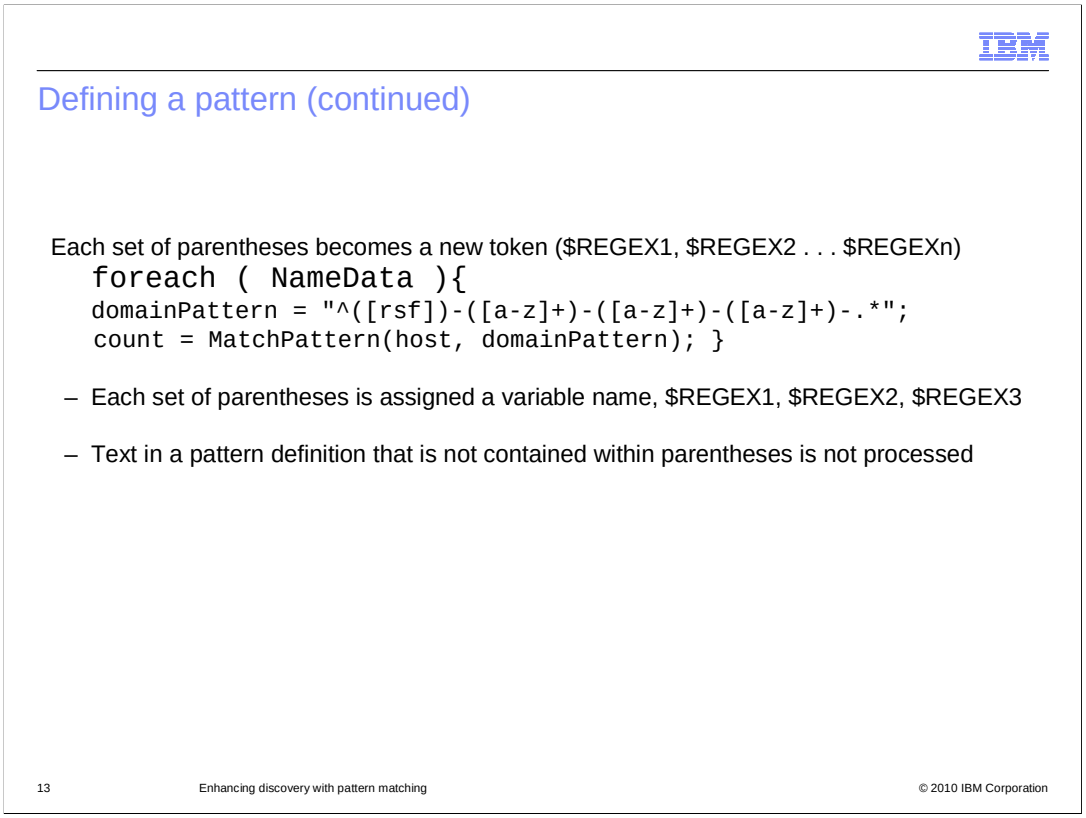
- Populate a RecordList called **NameData** with all the chassis node names by using a **RetrieveOQL** statement

```
// get all main nodes with DNS or system names
   RecordList NameData = RetrieveOQL("
   SELECT EntityName FROM         scratchTopology.entityByName
   WHERE EntityName LIKE '[a-zA-Z]' and
   (EntityType=1 OR EntityType=8);");
```


After initializing variables, the custom Host PatternMatching** stitcher must first get a list of all chassis devices. Chassis devices have an EntityType of 1 or 8. To get this list, use the **RecordList** keyword followed by the name of the list you want to create and then a query language statement to retrieve the data you want. This example also checks the **EntityName** to verify that the name contains alphabetic characters. This verification ensures that entities that have only an IP address for the **EntityName** will not be processed by this stitcher.



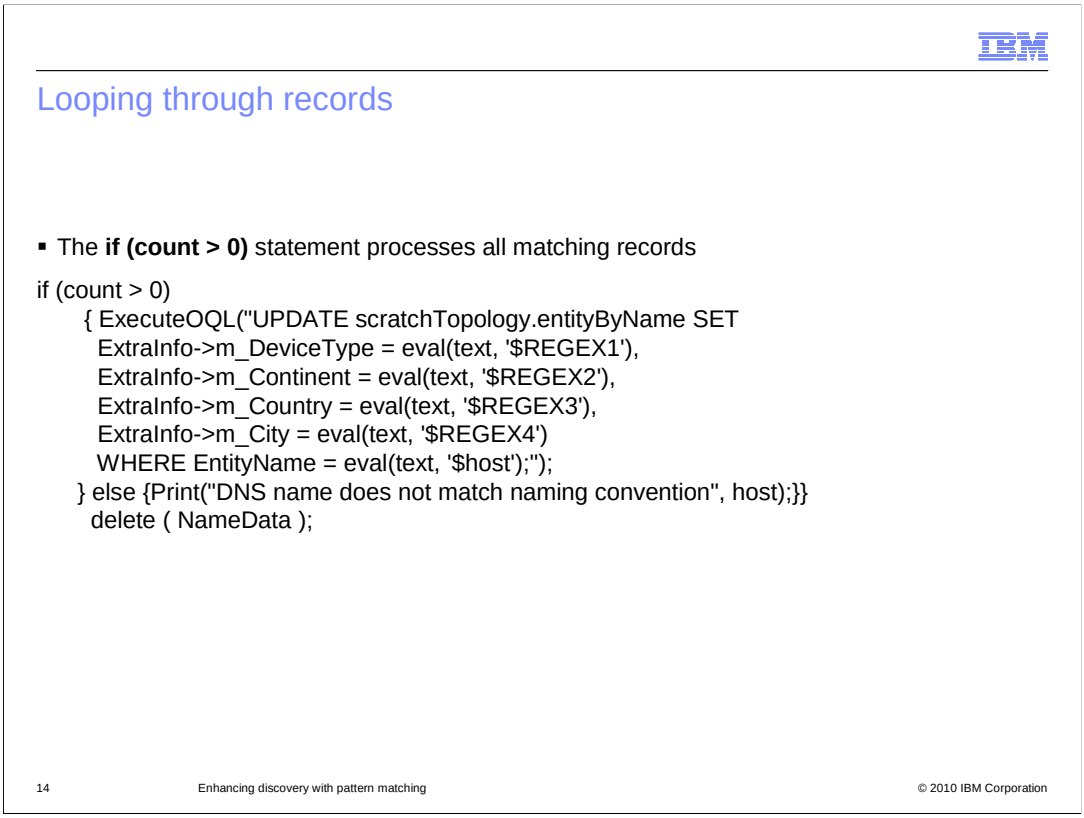
Because this example stitcher uses a name list, the stitcher must process each item in the list by looping through the list until all records have been evaluated. To do this, the stitcher uses a **foreach** statement**.** The stitcher will then compare each host name in the RecordList to the pattern that you specify. It returns a number of records in the recordlList that match the specified pattern. This count is decremented as each record is processed. After all records have been processed, the stitcher exits the foreach statement.



In the example shown here, the **foreach** statement loops through the names in the **NameData** record list. The statement evaluates each name in that list against the specified **domainPattern** variable. This evaluation establishes a regular expression value for each set of parentheses in the statement. The first set of parentheses is termed $REGEX1, the second $REGEX2, the third $REGEX3, and the fourth $REGEX4.

In this example, the hyphen is used to separate the various parts of the host name. The wildcard at the end of the **domainPattern** expression tells the stitcher that more characters will follow the last hyphen. However, because these characters do not appear inside of a set of parentheses in the definition of the domainPattern variable, they will be ignored.

The count statement is used here with the **MatchPattern** keyword to determine how many records in the record list match the specified pattern.



The sample code shown here uses an **if** statement to loop through the record list. This loop continues until all records have been processed. The query language function **UPDATE** modifies records in the scratch topology. Information that you are adding to discovery is typically added into subfields of the **ExtraInfo** field in the master.entityByName database.

This example also illustrates two best practices for custom stitchers. (1) A log message, using the Print statement, is created for each record that does not match the conditions of the stitcher code. You can review the log file and determine if any changes are needed to the stitcher code to properly process discovery data. (2) This code example ends with an important **delete** statement. Any time you create a record list, you are creating what a programmer calls an array. A record list or array must be deleted after it is no longer needed. This deletion frees up the memory that was used to store that list or array. A failure to do this results in a memory leak that can eventually cause problems for your system.

More information about how to create custom discovery stitchers is included in the workshop or advanced courses for IBM Tivoli Network Manager.

Calling the new stitcher

- Configure the PostScratchProcessing stitcher to run the new stitcher
  - Make a domain-specific copy of the PostScratchProcessing file
  - Insert the call for the new stitcher before the last two curly braces in **PostScratchProcessing.*domainName*.stch**

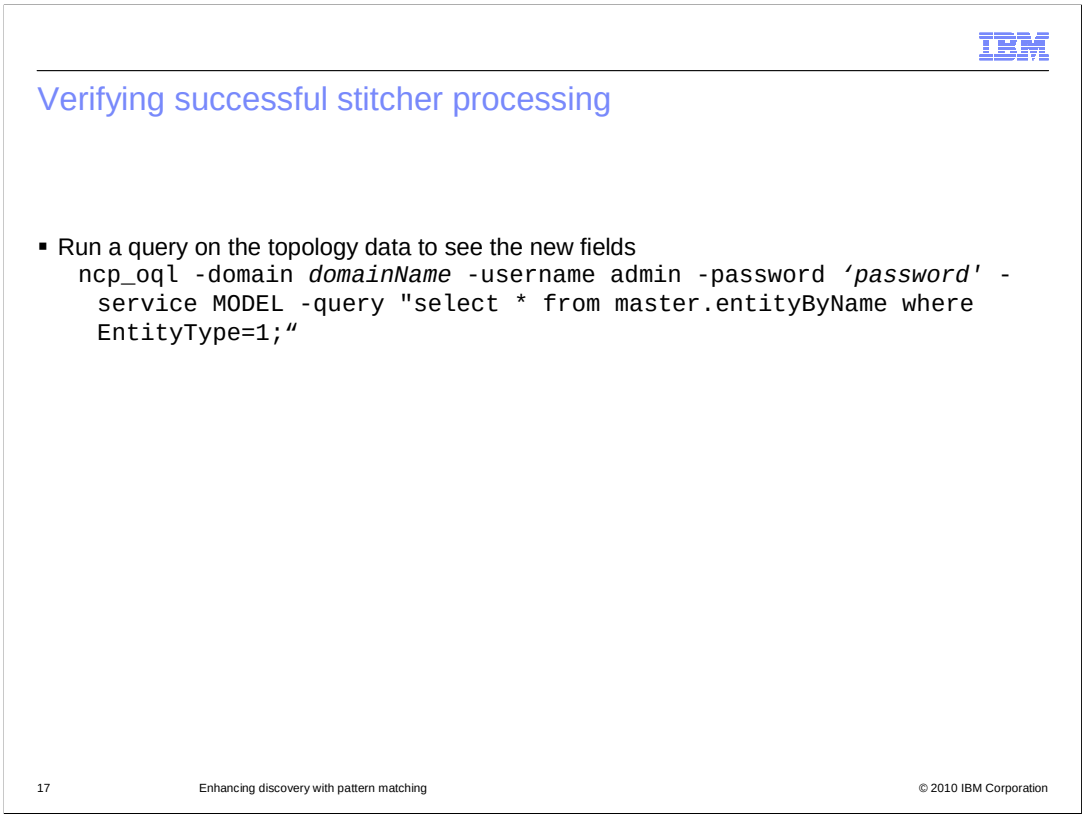
```
StitcherTimeCheck("AddLocationByIP","Beginning HostPatternMatching");
ExecuteStitcher('HostPatternMatching');
StitcherTimeCheck("HostPatternMatching","Beginning AddLookupInformation");
    }
}
```

15 Enhancing discovery with pattern matching

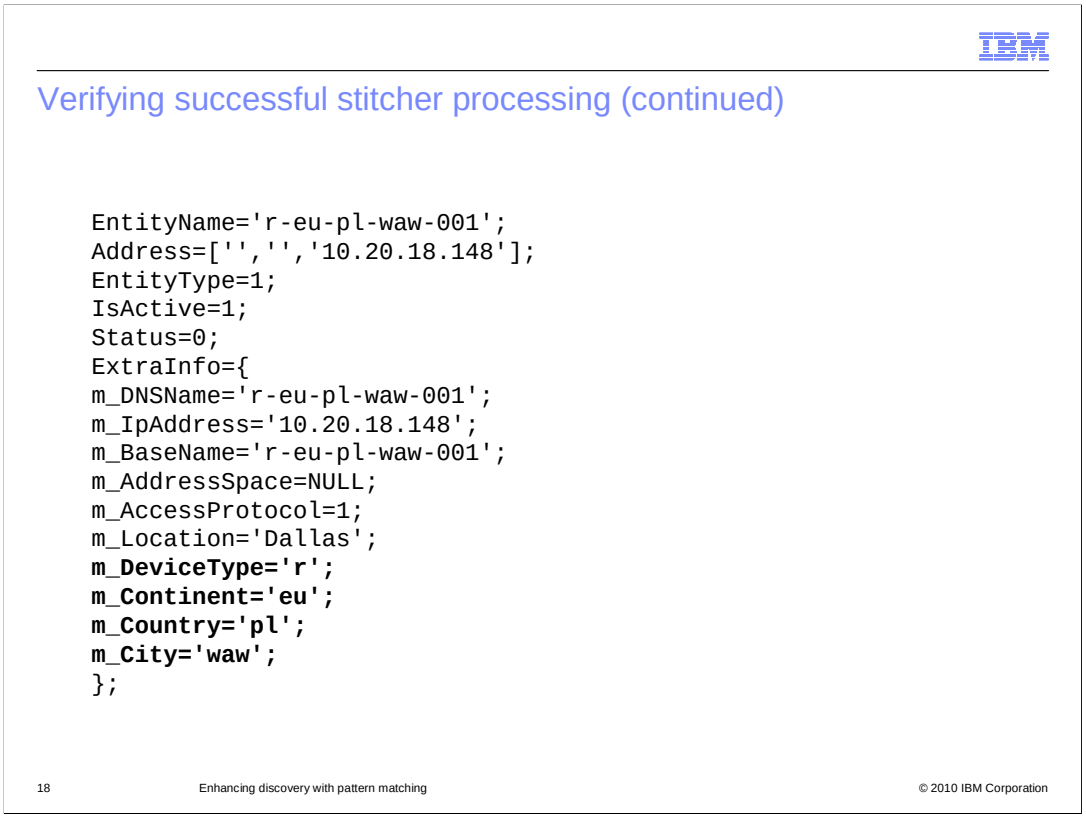
To run the new stitcher automatically during a typical discovery, the stitcher must be called from the PostScratchProcessing stitcher. Before modifying any installation-supplied IBM Tivoli Network Manager stitcher, always make a domain-specific copy of the stitcher. Edit only the domain-specific copy.

Go to the end of the stitcher file. Insert the **ExecuteStitcher** command to call the new custom stitcher near the end of the **PostScratchProcessing** stitcher file, just before the final two curly braces. You can also add a **StitcherTimeCheck** statement, which writes to the discovery log file, showing the completion of the previous stitcher and the beginning of your custom stitcher. By bracketing the command that runs your stitcher with time check statements, you can see error messages or messages indicating successful completion of your stitcher in the discovery log file.

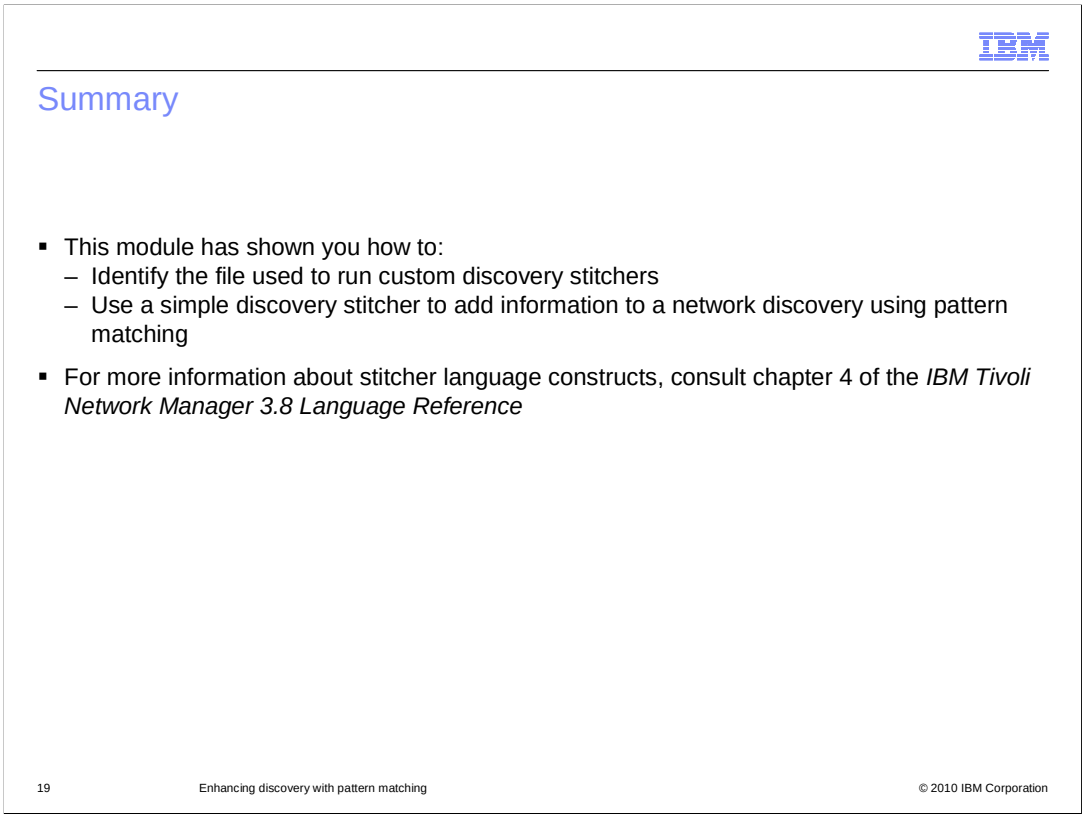
After the **PostScratchProcessing** stitcher has been modified and saved, you can verify successful stitcher processing in one of two ways. You can run a new full discovery of the network. However, if you have recently run another discovery, there is no need to find and interrogate all network devices again. Instead, you can run a query to restitch the network topology. Use a query like the example shown here, substituting your domain name and password. By inserting the value **CreateAndSendTopology** into the stitchers.actions table, that stitcher is run if the DISCO process is running.



After running a new discovery or restitching the network discovery, you can verify successful stitcher processing by running a query to see if the data is correctly populated. Run a query similar to the example shown here, substituting your correct domain name and password. This example query limits results to chassis entities because those entities are the only entities that are modified by the new custom **HostPatternMatching** stitcher.



The query returns results showing the newly added fields. In the example, you can see the device type, continent, country, and city information has been added for this chassis entity. If your query does not return successful results, check the discovery log file for messages about syntax errors. Correct those errors and rerun the **CreateAndSendTopology** stitcher.

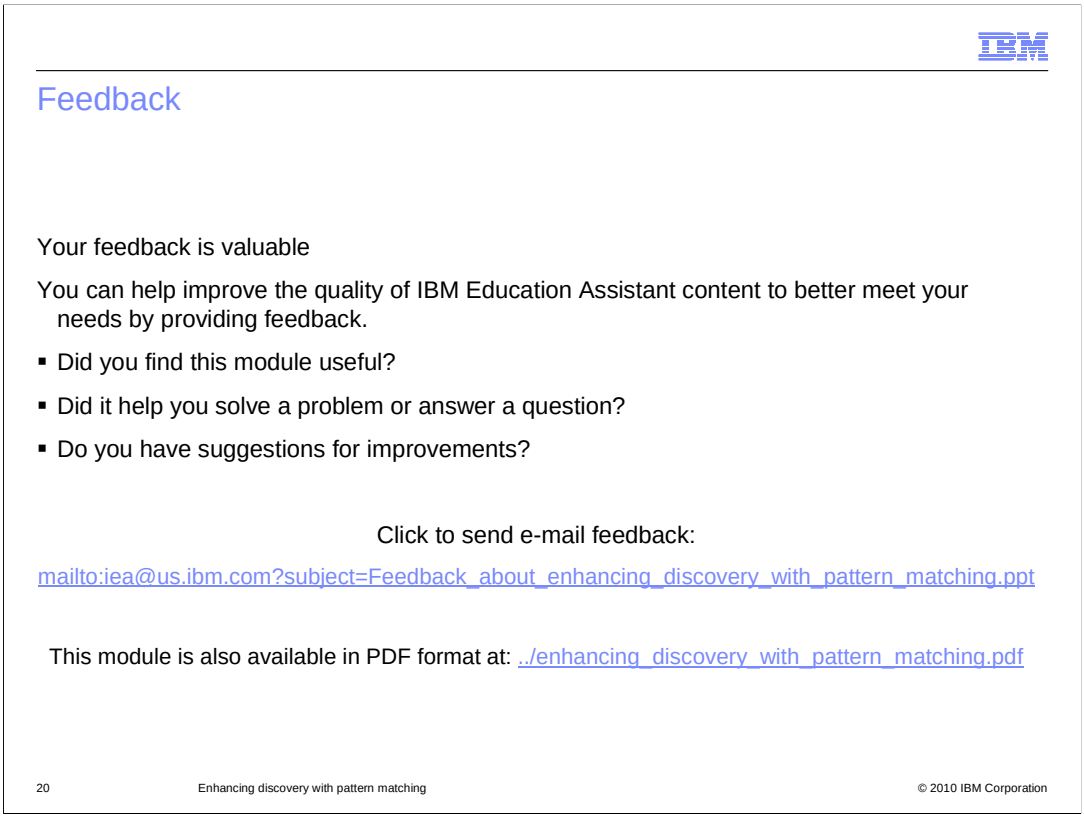


This module has shown you how to:

Identify the file used to run custom discovery stitchers

Use a simple discovery stitcher to add information to a network discovery

For more information about stitcher language constructs, consult chapter 4 of the IBM Tivoli Network Manager 3.8 Language Reference.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information