

# IBM Tivoli Network Manager IP Edition V3.8

## Enhancing discovery with pattern matching



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.

## Objectives

- 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

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

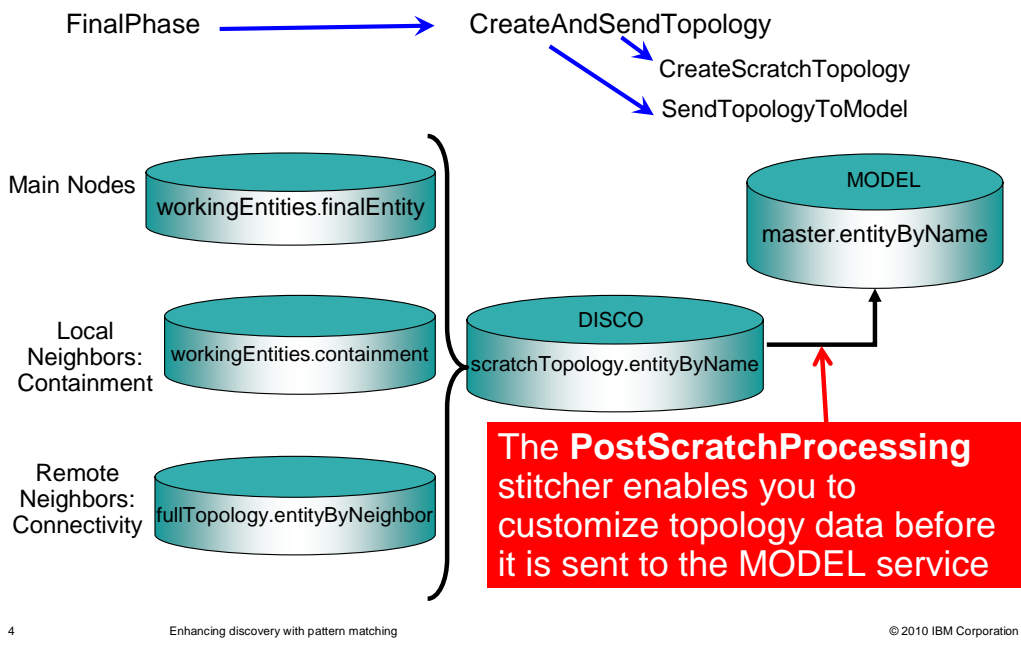
Identify the file that is used to run custom discovery stitchers

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

## ***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.

## Building the scratchTopology



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** sticher 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 sticher** is a final phase sticher 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** sticher. This sticher 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** sticher.

## ***Using pattern matching***

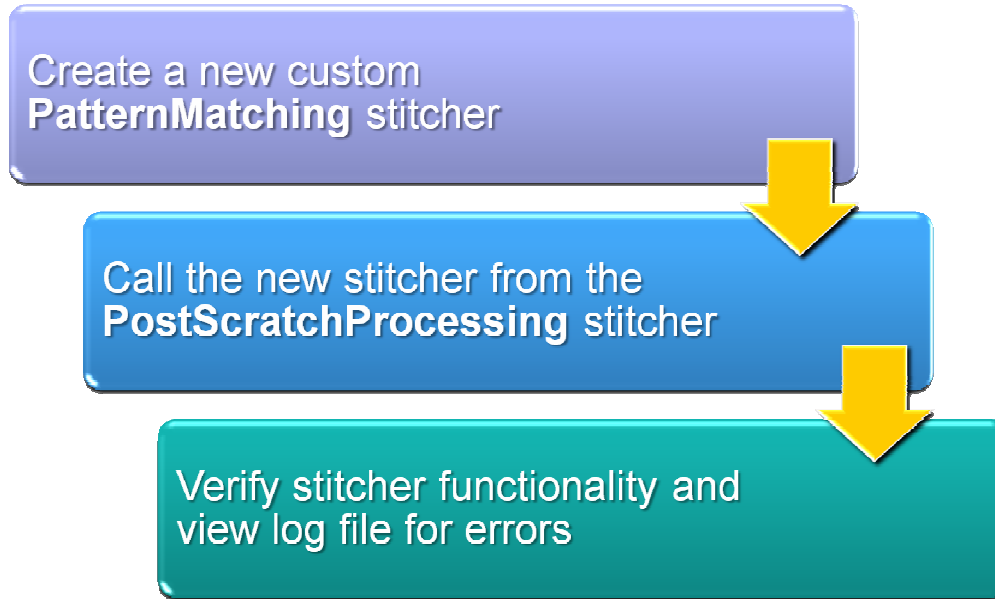
One example of enriching discovery data using a custom stitcher is to derive information using pattern matching on some field in the discovery data.

## 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.

## Solution workflow



7

Enhancing discovery with pattern matching

© 2010 IBM Corporation

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.

## ***Creating the custom pattern matching stitcher***

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 **HostPatternMatching** 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 **HostPatternMatching** sticher 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 sticher.

## Defining a pattern

- Use the **foreach** statement to process each record in NameData
- Declare and specify a pattern, such as domainPattern
- Determine the number of records that match the pattern
- Initialize a count to be used for looping through matching records until all records have been evaluated

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 recordList 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.

## Defining a pattern (continued)

Each set of parentheses becomes a new token (\$REGEX1, \$REGEX2 . . . \$REGEXn)

```
foreach ( NameData ) {  
  domainPattern = "^[rsf]-([a-z]+)-([a-z]+)-([a-z]+)-.*";  
  count = MatchPattern(host, domainPattern); }  
}
```

- Each set of parentheses is assigned a variable name, \$REGEX1, \$REGEX2, \$REGEX3
- Text in a pattern definition that is not contained within parentheses is not processed

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.

## Looping through records

- The **if (count > 0)** statement processes all matching records

```
if (count > 0)
{ ExecuteOQL("UPDATE scratchTopology.entityByName SET
  ExtralInfo->m_DeviceType = eval(text, '$REGEX1'),
  ExtralInfo->m_Continent = eval(text, '$REGEX2'),
  ExtralInfo->m_Country = eval(text, '$REGEX3'),
  ExtralInfo->m_City = eval(text, '$REGEX4')
  WHERE EntityName = eval(text, '$host');")
} else {Print("DNS name does not match naming convention", host);}
delete ( NameData );
```

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 **ExtralInfo** 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 sticher

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

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

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

Go to the end of the sticher file. Insert the **ExecuteStitcher** command to call the new custom sticher near the end of the **PostScratchProcessing** sticher 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 sticher and the beginning of your custom sticher. By bracketing the command that runs your sticher with time check statements, you can see error messages or messages indicating successful completion of your sticher in the discovery log file.

## Running the new sticher

- Run a new discovery
- Alternatively, you can run the **CreateAndSendTopology** sticher again manually

```
ncp_oql -domain domainName -username admin
        -password 'password' -service Disco -query "insert into
        stitchers.actions values ('CreateAndSendTopology');"
```

After the **PostScratchProcessing** sticher has been modified and saved, you can verify successful sticher 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 sticher is run if the DISCO process is running.



## Verifying successful stitcher processing

- Run a query on the topology data to see the new fields

```
ncp_oql -domain domainName -username admin -password 'password' -  
service MODEL -query "select * from master.entityByName where  
EntityType=1;"
```

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.

## Verifying successful stitcher processing (continued)

```
EntityName='r-eu-pl-waw-001';
Address=[' ',' ','10.20.18.148'];
EntityType=1;
IsActive=1;
Status=0;
ExtraInfo={
  m_DNSName='r-eu-pl-waw-001';
  m_IpAddress='10.20.18.148';
  m_BaseName='r-eu-pl-waw-001';
  m_AddressSpace=NULL;
  m_AccessProtocol=1;
  m_Location='Dallas';
  m_DeviceType='r';
  m_Continent='eu';
  m_Country='pl';
  m_City='waw';
};
```

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.

## Summary

- 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 using pattern matching
- For more information about stitcher language constructs, consult chapter 4 of the *IBM Tivoli Network Manager 3.8 Language Reference*

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.



## Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

[mailto:iea@us.ibm.com?subject=Feedback\\_about\\_enhancing\\_discovery\\_with\\_pattern\\_matching.ppt](mailto:iea@us.ibm.com?subject=Feedback_about_enhancing_discovery_with_pattern_matching.ppt)

This module is also available in PDF format at: [../enhancing\\_discovery\\_with\\_pattern\\_matching.pdf](http://../enhancing_discovery_with_pattern_matching.pdf)

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



## Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, and Tivoli are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2010. All rights reserved.