Network Manager IP Edition
Version 3 Release 9

Discovery Guide

SC27-2762-04

Network Manager IP Edition
Version 3 Release 9

Discovery Guide

SC27-2762-04

Note

Before using this information and the product it supports, read the information in “Notices” on page 367.

This edition applies to version 3.9 of IBM Tivoli Network Manager IP Edition (product number 5724-S45) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Appendix D. Helper System 339

Appendix E. Discovery stitchers . . . 341

Appendix F. Types of traps 361

Appendix G. Network Manager glossary 363

Notices 367

Index 371

Tables

About this publication

IBM Tivoli Network Manager IP Edition provides detailed network discovery, device monitoring, topology visualization, and root cause analysis (RCA) capabilities. Network Manager can be extensively customized and configured to manage different networks. Network Manager also provides extensive reporting features, and integration with other IBM products, such as IBM Tivoli Application Dependency Discovery Manager, IBM Tivoli Business Service Manager and IBM Systems Director.

The *IBM Tivoli Network Manager IP Edition Discovery Guide* describes how to administer and use Network Manager IP Edition to perform network discoveries.

Audience

This publication is for users and system and network administrators who configure IBM Tivoli Network Manager IP Edition.

IBM Tivoli Network Manager IP Edition works in conjunction with IBM Tivoli Netcool/OMNIbus; this publication assumes that you understand how IBM Tivoli Netcool/OMNIbus works. For more information on IBM Tivoli Netcool/OMNIbus, see the publications described in “Publications” on page x.

What this publication contains

This publication contains the following sections:

- Chapter 1, “About discovery,” on page 1

  Describes the concept of discovery, and the parameters that can be set to discover a network.
- Chapter 2, “Configuring network discovery,” on page 11

  Describes the prerequisites that must be met before a discovery can be configured and launched.

  Also describes how to run discoveries using:
  - The Discovery Wizard for performing an initial discovery, and for setting basic discovery parameters.
  - The Discovery Configuration GUI for setting advanced discovery parameters.
  - The CLI and the configuration files to configure the discovery process.

  There is also information on how to set complex discovery parameters, for example using Element Management Systems, MPLS and NAT.
- Chapter 3, “Monitoring network discoveries,” on page 131

  Describes how to monitor the state and progress of your network discovery using the GUI or the command line.
- Chapter 4, “Classifying network devices,” on page 147

  Describes how to change the way network devices are classified following discovery.
- Chapter 5, “Keeping discovered topology up-to-date,” on page 155

  Describes how to schedule a discovery, manually discover devices, and remove devices.
- “Troubleshooting discovery with reports” on page 163

Describes how to troubleshoot both the discovery process, and the network that you want to discover.

- Appendix A, “Discovery databases,” on page 183

  Describes the databases used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.
- Appendix B, “Discovery process,” on page 277

  Describes how IBM Tivoli Network Manager IP Edition produces a network topology that includes connectivity and containment data.
- Appendix C, “Discovery agents,” on page 303

  Describes the discovery agents available to run as part of your discovery. There is also guidance on the agents to select, based on the characteristics of your network.
- Appendix D, “Helper System,” on page 339

  Provides background information on the helpers, which are specialized applications that retrieve information from the network on demand.
- “Main discovery stitchers” on page 341

  Describes the stitchers supplied with IBM Tivoli Network Manager IP Edition.
- Appendix F, “Types of traps,” on page 361

  Describes the different types of traps that might be encountered by the Trap finder.

Publications

This section lists publications in the Network Manager library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

Your Network Manager library

The following documents are available in the Network Manager library:

- *IBM Tivoli Network Manager IP Edition Release Notes*, GI11-9354-00

  Gives important and late-breaking information about IBM Tivoli Network Manager IP Edition. This publication is for deployers and administrators, and should be read first.
- *IBM Tivoli Network Manager Getting Started Guide*, GI11-9353-00

  Describes how to set up IBM Tivoli Network Manager IP Edition after you have installed the product. This guide describes how to start the product, make sure it is running correctly, and discover the network. Getting a good network discovery is central to using Network Manager IP Edition successfully. This guide describes how to configure and monitor a first discovery, verify the results of the discovery, configure a production discovery, and how to keep the network topology up to date. Once you have an up-to-date network topology, this guide describes how to make the network topology available to Network Operators, and how to monitor the network. The essential tasks are covered in this short guide, with references to the more detailed, optional, or advanced tasks and reference material in the rest of the documentation set.
- *IBM Tivoli Network Manager IP Edition Product Overview*, GC27-2759-00

  Gives an overview of IBM Tivoli Network Manager IP Edition. It describes the product architecture, components and functionality. This publication is for anyone interested in IBM Tivoli Network Manager IP Edition.

- *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*, SC27-2760-00

  Describes how to install IBM Tivoli Network Manager IP Edition. It also describes necessary and optional post-installation configuration tasks. This publication is for administrators who need to install and set up IBM Tivoli Network Manager IP Edition.
- *IBM Tivoli Network Manager IP Edition Administration Guide*, SC27-2761-00

  Describes administration tasks for IBM Tivoli Network Manager IP Edition, such as how to administer processes, query databases and start and stop the product. This publication is for administrators who are responsible for the maintenance and availability of IBM Tivoli Network Manager IP Edition.
- *IBM Tivoli Network Manager IP Edition Discovery Guide*, SC27-2762-00

  Describes how to use IBM Tivoli Network Manager IP Edition to discover your network. This publication is for administrators who are responsible for configuring and running network discovery.
- *IBM Tivoli Network Manager IP Edition Event Management Guide*, SC27-2763-00

  Describes how to use IBM Tivoli Network Manager IP Edition to poll network devices, to configure the enrichment of events from network devices, and to manage plug-ins to the Tivoli Netcool/OMNIbus Event Gateway, including configuration of the RCA plug-in for root-cause analysis purposes. This publication is for administrators who are responsible for configuring and running network polling, event enrichment, root-cause analysis, and Event Gateway plug-ins.
- *IBM Tivoli Network Manager IP Edition Network Troubleshooting Guide*, GC27-2765-00

  Describes how to use IBM Tivoli Network Manager IP Edition to troubleshoot network problems identified by the product. This publication is for network operators who are responsible for identifying or resolving network problems.
- *IBM Tivoli Network Manager IP Edition Network Visualization Setup Guide*, SC27-2764-00

  Describes how to configure the IBM Tivoli Network Manager IP Edition network visualization tools to give your network operators a customized working environment. This publication is for product administrators or team leaders who are responsible for facilitating the work of network operators.
- *IBM Tivoli Network Manager IP Edition Management Database Reference*, SC27-2767-00

  Describes the schemas of the component databases in IBM Tivoli Network Manager IP Edition. This publication is for advanced users who need to query the component databases directly.
- *IBM Tivoli Network Manager IP Edition Topology Database Reference*, SC27-2766-00

  Describes the schemas of the database used for storing topology data in IBM Tivoli Network Manager IP Edition. This publication is for advanced users who need to query the topology database directly.
- *IBM Tivoli Network Manager IP Edition Language Reference*, SC27-2768-00

  Describes the system languages used by IBM Tivoli Network Manager IP Edition, such as the Stitcher language, and the Object Query Language. This publication is for advanced users who need to customize the operation of IBM Tivoli Network Manager IP Edition.
- *IBM Tivoli Network Manager IP Edition Perl API Guide*, SC27-2769-00

  Describes the Perl modules that allow developers to write custom applications that interact with the IBM Tivoli Network Manager IP Edition. Examples of custom applications that developers can write include Polling and Discovery Agents. This publication is for advanced Perl developers who need to write such custom applications.

- *IBM Tivoli Monitoring for Tivoli Network Manager IP User's Guide*, SC27-2770-00

  Provides information about installing and using IBM Tivoli Monitoring for IBM Tivoli Network Manager IP Edition. This publication is for system administrators who install and use IBM Tivoli Monitoring for IBM Tivoli Network Manager IP Edition to monitor and manage IBM Tivoli Network Manager IP Edition resources.

Prerequisite publications

To use the information in this publication effectively, you must have some prerequisite knowledge, which you can obtain from the following publications:

- *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*, SC23-9680

  Includes installation and upgrade procedures for Tivoli Netcool/OMNIbus, and describes how to configure security and component communications. The publication also includes examples of Tivoli Netcool/OMNIbus architectures and describes how to implement them.

- *IBM Tivoli Netcool/OMNIbus User's Guide*, SC23-9683

  Provides an overview of the desktop tools and describes the operator tasks related to event management using these tools.

- *IBM Tivoli Netcool/OMNIbus Administration Guide*, SC23-9681

  Describes how to perform administrative tasks using the Tivoli Netcool/OMNIbus Administrator GUI, command-line tools, and process control. The publication also contains descriptions and examples of ObjectServer SQL syntax and automations.

- *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*, SC23-9684

  Contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands.

- *IBM Tivoli Netcool/OMNIbus Web GUI Administration and User's Guide* SC23-9682

  Describes how to perform administrative and event visualization tasks using the Tivoli Netcool/OMNIbus Web GUI.

Accessing terminology online

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

http://www.ibm.com/software/globalization/terminology

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File** > **Print** window that allows your PDF reading application to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to the following Web site:

   http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss
2. Select your country from the list and click **Go**. The Welcome to the IBM Publications Center page is displayed for your country.
3. On the left side of the page, click **About this site** to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully.

Accessibility features

The following list includes the major accessibility features in Network Manager:

- The console-based installer supports keyboard-only operation.
- The console-based installer supports screen reader use.
- Network Manager provides the following features suitable for low vision users:
  - All non-text content used in the GUI has associated alternative text.
  - Low-vision users can adjust the system display settings, including high contrast mode, and can control the font sizes using the browser settings.
  - Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.
- Network Manager provides the following features suitable for photosensitive epileptic users:
  - Web pages do not contain anything that flashes more than two times in any one second period.

The Network Manager Information Center, and its related publications, are accessibility-enabled. The accessibility features of the information center are described in Accessibility and keyboard shortcuts in the information center.

Extra steps to configure Internet Explorer for accessibility

If you are using Internet Explorer as your web browser, you might need to perform extra configuration steps to enable accessibility features.

To enable high contrast mode, complete the following steps:

1. Click **Tools** > **Internet Options** > **Accessibility**.
2. Select all the check boxes in the Formatting section.

If clicking **View** > **Text Size** > **Largest** does not increase the font size, click **Ctrl +** and **Ctrl -**.

IBM® and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

http://www.ibm.com/software/tivoli/education

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Go to the IBM Software Support site at http://www.ibm.com/software/support/probsub.html and follow the instructions.

IBM Support Assistant

The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to http://www.ibm.com/software/support/isa

Conventions used in this publication

This publication uses several conventions for special terms and actions and operating system-dependent commands and paths.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:** and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a *point-to-point* line)

- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data
- Variables and values you must provide: ... where *myname* represents....

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses environment variables without platform-specific prefixes and suffixes, unless the command applies only to specific platforms. For example, the directory where the Network Manager core components are installed is represented as NCHOME.

When using the Windows command line, preface and suffix environment variables with the percentage sign **%**, and replace each forward slash (**/**) with a backslash (****) in directory paths. For example, on Windows systems, NCHOME is %NCHOME%.

On UNIX systems, preface environment variables with the dollar sign **$**. For example, on UNIX, NCHOME is $NCHOME.

The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. About discovery

Configure discovery by setting the parameters that govern how the discovery is performed.

About types of discovery

Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Discovery and rediscovery

Discovery

The term discovery is used generally to mean any kind of discovery. Technically, only the first discovery that is run after the discovery engine, ncp_disco, is started can properly be called a discovery, and every discovery after that is a rediscovery. Because there is no discovery data in memory yet, discoveries take slightly longer than rediscoveries.

Rediscovery

After a discovery has been run, any further discoveries that are run are rediscoveries. Rediscoveries use a different data flow to discoveries, with some different stitchers and databases. If ncp_disco is restarted, the next discovery is again just a discovery, and further discoveries after that are rediscoveries. Unless you are running advanced discoveries or modifying the discovery data flow, the difference between a discovery and a rediscovery is not usually important, and, for ease of reading, the instructions in this information do not distinguish between discovery and rediscovery unless it is necessary.

Full and partial discovery

Full discovery

A full discovery is a discovery run with a large scope, intended to discover all of the network devices that you want to manage. Full discoveries are usually just called discoveries, unless they are being contrasted with partial discoveries.

Partial discovery

A partial discovery is a subsequent rediscovery of a section of the previously discovered network. The section of the network is usually defined using a discovery scope consisting of either an address range, a single device, or a group of devices. A partial discovery relies on the results of the last full discovery, and can only be run if the discovery engine, the `ncp_disco` process, has not been stopped since the last full discovery. A partial discovery is, therefore, actually a type of rediscovery.

Automatic and scheduled discovery

You can run discoveries on demand, using the wizard, the GUI, or the command line. You can also configure discovery to start automatically.

Automatic discovery
After a discovery has finished, the discovery process enters a reactive state, known as rediscovery mode, in which another discovery can be triggered automatically by receipt of a trap from a network device.

Scheduled discovery
You can schedule a discovery to start at a certain time.

Related concepts:

"Full and partial rediscovery" on page 299
By modifying the stitchers, you can configure the way DISCO treats devices that are found in the rediscovery mode.

Related tasks:

"Scheduling a discovery" on page 155
After a full discovery has completed, you can schedule further discoveries by editing the `FullDiscovery.stch` file.

"Starting a discovery" on page 43
After you configure a discovery, you can start and, if necessary, stop the discovery.

"Starting partial discovery from the GUI" on page 158
Starting a partial discovery involves defining a seed and scopes.

"Configuring automatic discovery" on page 155
Network Manager provides a mechanism to automatically trigger a partial discovery based on receipt of a trap. This is performed by the Disco plug-in to the Event Gateway. Device traps might indicate a change in a network device or the presence of a new network device.For more information on the Disco plug-in, see the *IBM Tivoli Network Manager IP Edition Event Management Guide*.

Scopes

Define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude. The areas of the network to be included in the discovery process, or excluded from the discovery process are collectively known as the discovery scope.

There are several benefits to limiting the discovery scope:

- It is important to limit discovery scope because the range of IP addresses considered by the default discovery process is potentially unlimited. Without a scope, the discovery attempts to recognize every network device. By limiting the scope, you can concentrate on the important areas of your network.

  Attention: If there are routes out of your network to the Internet, then an unscoped discovery will find these routes and proceed to discover parts of the Internet.
- You might also want to restrict the scope of the discovery to control the discovery of sensitive devices that you do not want to poll. A device might be considered sensitive because there is a security risk involved in polling the device, or because polling might overload the device. You can specify that particular devices are discovered but not instantiated to an AOC definition (such devices are discovered but are *not* represented in the network topology and their details are *not* sent to MODEL). You can also restrict devices from being discovered (SNMP access to any such device is not attempted).
- Another reason for scoping the discovery is that it restricts the amount of data Network Manager tries to download from the routing tables of individual routers. Without this restriction, if Network Manager finds a router that knows the routing table for the whole Internet, then discovery takes a very long time to complete.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not aupport an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead this address must be entered as `::ffff:c000:280` (standard colon-separated IPv6 format).

Types of scoping

Network Manager offers several types of scoping.

You can enable the following types of scoping:

- You can include or exclude areas of your network (either subnet ranges or specific devices) in the discovery. Each configured area is referred to as a *zone*.

  Tip: If your subnet is sparsely populated, including individual routers is likely to result in a faster discovery than including the whole subnet.
- Zones can be specified within zones: within a given inclusion zone, you can specify devices or subnets that are not to be detected. These devices are not pinged by the Ping finder or interrogated by the discovery agents. For example, you can define an include scope zone consisting of the Class B subnet 1.2.0.0/16, and within that zone you can specify an exclude scope zone consisting of the Class C subnet 1.2.3.0/24. Finally, within the exclude scope zone you could specify an include scope zone 1.2.3.128/26.
- You can configure a filter that determines whether a discovered device is interrogated for connectivity information.
- You can configure a filter that determines whether a device within a defined zone is to be instantiated. If a device is instantiated, it is displayed on the network map. Devices that are not instantiated are not sent to MODEL.
- You can configure multicast scoping. This enables you to configure which multicast subnets to use as scopes for your multicast discovery.

Defining discovery zones to restrict discovery

To restrict the discovery, you must define discovery zones. You can define discovery zones in several ways.

Choose one of the following methods to define a discovery zone:

- Define discovery zones using the Discovery Configuration GUI.
- Construct zones by appending an OQL insert into the `scope.zones` table with the `DiscoScope.cfg` configuration file. This method is for more experienced users.

Note: If nothing is specified in the `scope.zones` table then everything is considered to be in scope.

For each zone you must specify the following information:

- The type of network protocol used by the zone, although currently only IP is supported. You can define as many zones as necessary. Multiple zones can also be defined within the same insert.
- The action to be taken for the zone, where `m_action=1` means include in the discovery, and `m_action=2` means exclude. You can define both inclusion and exclusion zones. The action to be taken in the smallest zone overrides the actions in the larger zones.
- A list of varbinds (`name=value`) that define the present discovery zone.

Related tasks:

"Defining multiple inclusion zones" on page 19
You can define multiple inclusion zones in the scope.zones table.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Seeds

Configure seeds to specify the locations from which to begin discovering devices. Discovery seeds can be IP addresses, or subnet addresses.

You can specify seeds in several ways:

- Using the Ping finder: You specify the IP addresses or subnet addresses to discover first.
- Using the File finder: You provide one or more files that each contain a list of IP addresses or subnet addresses.

Tip: To restrict discovery to a list of specific devices, seed the discovery with a list of devices using the File finder or the Ping finder, and disable feedback in the **Advanced** tab of the Network Discovery Configuration GUI.

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

Device access

Configure device access by specifying SNMP community strings and Telnet parameters so that the system can access network devices.

Configure device access as follows:

- Specify SNMP community strings so that Network Manager can access and interrogate the network devices that use SNMP. Network Manager supports SNMP v1, v2, and v3,
- Specify Telnet parameters so that Network Manager can access and interrogate the network devices that use Telnet.

Agents

Use discovery agents to retrieve information about devices on the network. Select the right agents for your discovery depending on your network type.

Discovery agents retrieve device details and investigate device connectivity. They can also report the existence of new devices by finding new connections when investigating device connectivity. The discovery agents can be used for specialized tasks. For example, the ARP Cache discovery agent populates the Helper Server database with IP address to MAC address mappings.

Default agents are provided for the type of discovery you want to perform, for example a layer 2 or layer 3 discovery. You can select different sets of agents for full discoveries and for partial discoveries. The agents vary because connectivity information varies with the technology of the hardware in the network.

Filters

Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

After you have defined the scope of your discovery using the **Scope** tab, it is possible to restrict the scope using filters. For example, you might want to maintain the scope zones that you defined earlier but restrict the scope based on location (for example, New York hardware only) or based on hardware supplier (for example, Cisco devices only).

You can filter out devices based on a variety of criteria, including location, technology, and manufacturer.

By default, the discovery filters do not filter out the Network Manager host because it usually also serves as the polling station for root cause analysis. For root cause analysis to work correctly, the polling station, and hence the Network Manager host machine, must be part of the topology.

For more information on root cause analysis, see the *IBM Tivoli Network Manager IP Edition Administration Guide* and the *IBM Tivoli Network Manager IP Edition Network Troubleshooting Guide*.

If you do need to filter out the Network Manager host, then you need to modify the following stitchers and remove the sections of code, indicated by comments, that prevent the Network Manager host from being filtered. The stitchers are DetectionFilter.stch and InstantiationFilter.stch.

Prediscovery filter

You might want to apply this filter to sensitive devices that you do not want to poll. A device might be considered sensitive because there is a security risk involved in polling the device, or because polling might cause the device to overload.

Prediscovery filters prevent the discovery from retrieving detailed data or connectivity data from the device and prevent discovered devices from being polled for connectivity information. Only devices matching the prediscovery filter are fully discovered. If no prediscovery filter is defined, then all devices within the scope are discovered.

Prediscovery filters provide a mechanism to base discovery on complex IP ranges that cannot be easily defined in the **Scope** tab. It can be used to filter out devices based on their sysObjectId value. Default filters exist to filter out end nodes, printers, and similar devices. You can create quite complex multiple filters, which makes this feature very powerful but try to ensure that filters are designed so that they can be easily maintained. The filter acts on the fields of the `details.returns` OQL table in the discovery (Disco) service, so you can use fields other than IP addresses, such as m_ObjectId (equivalent to sysObjectId). A device must pass all filters to be discovered.

Important: Design the filter logic so that you do not need to modify the prediscovery filters every time you add new scopes.

You can configure the filter condition to test against any of the columns in the Details.returns table. But, you might need to use the IP address (m_UniqueAddress) as the basis for the filter to restrict the detection of a particular device. If the device does not grant SNMP access to the Details agent, the Details agent might not be able to retrieve MIB variables such as the Object ID. However, you are guaranteed the return of at least the IP address when the device is detected.

You can define multiple prediscovery filters. Filters are combined automatically using a Boolean AND expression. All criteria defined in all filters must be matched.

Post-discovery filter

You might want to apply this filter to devices that you do not want to poll, such as workstations and printers. A post-discovery filter restricts device instantiation. If a post-discovery filter is defined, only devices that pass the criteria are instantiated, that is, sent to MODEL. If no post-discovery filter is defined, then all discovered devices are sent to MODEL.

Data on unclassified devices is held in the NCIM topology database; however, the device cannot be visualized in Topoviz and cannot be polled.

You can define multiple post-discovery filters. Filters are combined automatically using a Boolean AND expression, which means that all criteria defined in all filters must be matched.

The post-discovery filter operates on the scratchTopology.entityByName table. Hence, the fields available in this filter are different from those available to the prediscovery filter. The post-discovery filter operates on topology fields rather than on basic device information.

Related concepts:

"Creating the topology" on page 292
The creation of the topology is carried out in several steps.

Related tasks:

"Setting discovery filters" on page 28
Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Related reference:

"Main discovery stitchers" on page 341
This topic lists all discovery stitchers.

Appendix A, "Discovery databases," on page 183
There are various specialized databases that are used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.

"scratchTopology database schema" on page 263
The scratchTopology database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table name is: scratchTopology.entityByName.

Domain Name System

Configure DNS to enable the discovery to access DNS services that are used to perform domain name lookups.

You can configure three types of Domain Name System:

DNS server
A server on the network that is dedicated to performing domain name resolution.

File
The name of a file held on the Network Manager host that contains IP addresses and host names in lookup table format.

System
The local DNS system on the Network Manager machine.

Network address translation

Configure data for NAT gateways in your network.

NAT gateways provide mappings between private IP address in your network and public device IP addresses. You can enable the system to discover devices within private address spaces by configuring data for NAT gateways.

Advanced settings

Configure advanced discovery settings to increase the speed of the discovery, and balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server. Advanced settings control features of the discovery such as concurrent processes and timeouts.

Note: Modify the advanced settings only if you are an experienced Network Manager user.

You can configure the following advanced discovery settings:

Finder parameters:
Finders are discovery subsystems that discover devices on the network. You can configure parameters such as timeouts, number of retries, and number of threads for the finders.

Helper parameters
Helpers are discovery applications used by agents to retrieve information from devices. You can configure parameters such as timeouts, number of retries, and number of threads for the helpers.

Other parameters
You can configure complex discovery settings, such as enabling caching of discovery tables, VLAN modeling, discovery failover, File finder verification, and parameters that affect the speed of partial discovery.

Most of the advanced discovery parameters are optional.

Context-sensitive discovery

If you need to discover devices such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

In a context-sensitive discovery, information about a device is passed from the `returns` table of the Details agent to the `despatch` table of the relevant Context agent.

The Context agents use the filters in the .agent files of the agents to determine which devices to process. This is true for all discovery agents. If the device is not of a type which supports virtual routers, that is, does not need context-sensitive processing, it is passed directly to the Associated Address agent.

Related concepts:

"Discovering device details (context-sensitive)" on page 288
The discovery of context-sensitive device details is carried out in several steps.

Related reference:

"Context-sensitive discovery agents" on page 329
There are several agents that take part in a context-sensitive discovery.

Helpers

The helpers are specialized applications that retrieve information from the network on demand. The default helper configuration is sufficient for most networks. However, you might decide to alter the configuration for several reasons.

Configuring the Helper System can speed up network discovery, but is recommended for experienced users.

Although the discovery agents retrieve connectivity information, they do not have any direct interaction with the network. Instead, they retrieve connectivity information through the Helper System, which consists of a Helper Server and various helpers.

Reasons to configure the helpers include:

- To speed up the discovery process, you could reduce the helper timeouts and number of retries.
- If you have a very reliable network in which devices respond quickly, you can specify a small default timeout.
- You might want to change the default timeouts for the SNMP and Telnet helpers if you have many devices that either do not respond to SNMP and Telnet or that are set up not to respond to Telnet or SNMP access. A large default timeout would therefore mean that the helpers wait for a long time for responses they never receive.
- To reduce the amount of network traffic caused by a discovery, you could increase the timeout and disable broadcast and multicast pinging.

Specialized discoveries

You can configure the system to perform more complex discoveries, such as Multiprotocol Label Switching (MPLS) and Network Address Translation (NAT) discovery.

Specialized discoveries include:

Element Management System (EMS) discoveries
Collect topology data from Element Management Systems and integrates this data into the discovered topology.

MPLS discoveries
Discover layer 3 virtual private networks (VPNs) and enhanced layer 2 VPNs running across MPLS core networks.

NAT discoveries
Discover NAT gateway devices to retrieve data on devices in private address spaces.

Third-party discoveries:
Discover intervening provider networks as a "third-party" object on multiple networks that run across a provider network (for example, enterprise VPNs across a provider MPLS core network).

Chapter 2. Configuring network discovery

Configure how your network is discovered, including which kinds of devices you want to discover, and where the boundaries of the discovery should be.

Network Manager provides tools for discovering your network using a phased approach.

- Use the Discovery Configuration Wizard to perform initial discoveries. The wizard provides a guided discovery and makes configuration choices for you based on the answers that you provide as you work through the wizard.
- Use the Discovery Configuration GUI to perform subsequent discoveries. Using the GUI you can configure detailed discovery settings, including scope, seeds, community strings, agent selection and many other configuration settings.

Note: You can also configure a discovery using the discovery configuration files and the command line. However, you should configure discovery this way only if you are an experienced Network Manager user and you understand the different aspects of discovery, including the discovery processes, phases, stages, Helpers, agents, stitchers and traps.

For information on how to manually edit a discovered topology following discovery, see the *IBM Tivoli Network Manager IP Edition Network Visualization Setup Guide*.

Planning for discovery

Before configuring and running a discovery, you should check several system settings, parameters, and requirements.

The following notes help you plan for the discovery.

Saving changes in the Network Discovery Configuration GUI

To save configuration changes that you have made during a session, remember to click the **Save** button before you log out, close the browser window, or close the Network Discovery Configuration tab. It is good practice to click **Save** as you move from tab to tab.

Operating system

Ensure that the host on which Network Manager is running is fully patched with the latest patches.

Discovery scope

Consider the following questions and points related to the discovery scope:

- Is the positioning of the Network Manager host within the network?
- Is the host positioned to interrogate all devices that you want to include in your discovery?
- Consider all necessary networks, subnetworks and determine the associated netmasks.
- Are there any parts of the network that you want to exclude?
- Gather all relevant SNMP community strings for the devices within the scope

Routing

Ensure that each of the networks and subnetworks to be discovered is reachable using the ICMP process. If necessary, add routes to Network Manager host machine using the **route add** command.

Access control lists

Network Manager uses several protocols that might need to pass through firewalls. These protocols are ICMP, SNMP, DNS, ARP, SSH, and TELNET. To ensure that Network Manager can access devices behind these firewalls, advise the relevant firewall administrators to prepare the firewalls.

Root cause analysis

To perform root cause analysis on devices within a topology, the discovery must identify all the devices on which you might want to perform root cause analysis. In addition, the discovery must identify the Network Manager polling station. For more information on root cause analysis, see the *Network Manager Monitoring and RCA Guide*.

Discovering the network using the wizard

The discovery configuration wizard is for users who have limited experience in configuring discoveries.

Important: If you want to keep discovery configuration settings that you made previously using the GUI, do not use the wizard. The discovery configuration wizard overwrites all previous settings.

Related tasks:

"Monitoring network discovery from the GUI" on page 131
From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

Launching the wizard

Select a domain and launch the wizard to start configuring and running a discovery.

To launch the wizard, complete these steps.

1. Click **Discovery** > **Network Discovery Configuration**.
2. At the top left of the Network Discovery Configuration tab, select the domain in which you want to run a discovery from the **Domain** menu.
3. Click the wizard button to the right of the **Domain** menu.

Choosing a scoped or unscoped discovery

The Discovery Scope window provides the option of a scoped or unscoped discovery.

To choose a scoped or unscoped discovery, complete these steps.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not aupport an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

1. Select **Scoped** or **Unscoped**.

Scoped

A scoped discovery restricts the discovery to a certain part of your network. To specify a scoped discovery, tell the wizard which area of the network to restrict the discovery to, and assign IP addresses or subnets as seeds to ping to begin the discovery.

Unscoped

An unscoped discovery attempts to discover your entire network. However, you still need to assign IP addresses or subnets as seeds to ping to begin the discovery.

Attention: If there are routes out of your network to the Internet, an unscoped discovery will find these routes and start to discover parts of the Internet.

2. If you selected **Scoped**, specify which area of the network to which to restrict the discovery.

   Specify one or more subnets to use for both scoping and seeding by clicking **New** and typing an IP address and a netmask.

   Restriction: For performance reasons, only IPV4 addresses will be pinged. To ping IPV6 addresses use the Seed tab in the Discovery Configuration GUI.
3. If you selected the **Unscoped** option, specify the seeds to use for your unscoped discovery.

   Click **New...** and specify one or more IP addresses.

Configuring SNMP access using the wizard

Specify address-specific, network-specific, or global community strings on the SNMP Community Strings window.

For SNMP version 3, you can also specify passwords for community strings.

When discovering devices using SNMPv3, the Cisco switches must have the VLAN context added to the view group for each VLAN.

To configure SNMP access, complete the following steps.

1. For each SNMP community string and associated password that you want to define:
   a. Click the **New** icon above the **SNMP Community Strings** table to display the SNMP Password Properties window.
   b. Specify address-specific, subnet-specific, or global SNMP community strings, and supply passwords for these community strings in the case of SNMPv3.

      You might need to enter a community string more than once. For example, enter a string for SNMPv1, enter another string for SNMPv2, and another string for SNMPv3.

      Specifying community strings by subnets results in a more efficient and faster discovery.

      Restriction: It is best practice not to use the at symbol (@) in community strings. Using this symbol in a community string can cause problems connecting to devices at discovery time.

2. Use the up and down arrow keys to put the community strings in order of most frequently expected use. Put more frequently used community strings at the top.

Configuring Telnet access using the wizard

On the Telnet Access window, set the Telnet access parameters.

To configure Telnet access, complete these steps.

1. After you have specified SNMP community strings, click the **New** icon on the Telnet Access window.
2. For each set of Telnet-accessible devices for which you want to define prompts and passwords, click **New**.
3. On the Telnet Passwords window, specify a set of Telnet-accessible devices (all devices, all devices within a specified subnet or a single IP address) together with prompts, login IDs, and login passwords for this set of devices.

Specifying type of discovery

On the Discovery Type window, specify the type of discovery: a Layer 3 or a Layer 2 discovery.

A Layer 3 discovery is quicker, but the results of a Layer 3 discovery cannot be used for root cause analysis. A Layer 2 discovery is more detailed and the results can be used for root cause analysis.

To specify discovery type, complete these steps.

1. On the Discovery Type window, specify a Layer 2 or Layer 3 discovery.
2. If you selected **Layer 3**, the End Node Discovery window is displayed.

   On the End Node Discovery window, you can filter out end node devices such as workstations and printers. You can also filter out devices with no SNMP access.

   Tip: Filtering out all end nodes in networks that have large numbers of end nodes can lead to speed and performance improvements in your discovery.
3. If you selected **Layer 2 and Layer 3**, the VLAN Modelling window is displayed.

   In the VLAN Modelling window, you configure the discovery to model VLANs in the resulting topology. This enables VLANs to be considered when performing root cause analysis. VLANs are a Layer 2 concept and VLAN modelling is required for Layer 2 discoveries only. Specify whether you want to model VLANs. When you have specified an option, click **Next** to display the End Node Discovery window.

Optimizing the discovery

On the Discovery Optimization window, optimize the discovery for connectivity, richness of information, and speed.

To optimize the discovery, complete these steps.

1. Provide varying amounts of connectivity information by selecting one of these options.

   Best possible connectivity accuracy and richness of information
   This option provides comprehensive connectivity information between switches, end nodes and routers as well as detailed information on each device discovered. However, the discovery might require a significant amount of time to complete.

   Best possible connectivity accuracy but prefer speed to richness of information
   This option provides comprehensive connectivity information. However, the discovery provides less detailed information on each device discovered in order to provide a faster discovery.

   Rich device information but prefer speed to accurate connectivity
   This option provides detailed information on each device discovered. However, the discovery provides less detailed connectivity information to provide a faster discovery. For example, the discovery might provide information on how switches are connected to each other, but it might not provide information on connectivity between switches and end nodes or between switches and routers.

   Note: This option is more suitable if you want to gather inventory data instead of perform root cause analysis (RCA). RCA is dependent on accurate connectivity data.

   Fastest discovery time
   This option focuses on the speed of the discovery. However, limited connectivity information is provided as well as less detailed information on each single device.

2. If you select either of the first two options, this means that accurate connectivity is important. The Network Reliability window is displayed.
3. If you select either of the last two options, this means that you are willing to compromise on the accuracy of connectivity information to ensure a faster discovery. In this case, the wizard asks how much of your network is made up of Cisco devices. If a large proportion of the network is made up of Cisco devices, then the wizard can switch off agents that discover connectivity for non-Cisco devices, thereby speeding up the discovery significantly. The Cisco Hardware window is displayed.

   a. Specify how much of your network is made up of Cisco hardware by selecting one of these options.

      All of it
      This option directs the wizard to run the Cisco Discovery Protocol (CDP).

      Most of it, Some of it, Don't know
      This option directs the wizard to run the CDP. However, if you chose a Layer 2 and Layer 3 discovery or if you indicated that you want to exclude end nodes from the discovery, then this option invokes the Spanning Tree Protocol (STP) as well as CDP.

None of it
This option specifies that neither the CDP nor the STP protocol is used.

b. When you have selected one of these options, click **Next**.

c. If your response to the Cisco hardware question was **All of it** or **None of it**, then the Network Reliability window is displayed.

d. If your response to the Cisco hardware question was **Most of it**, **Some of it**, or **Don't know**, then the Spanning Tree Protocol window is displayed, on which you specify whether the spanning tree protocol is enabled on all network switches.

Indicating the reliability of your network

On the Network Reliability window, select a description of the reliability of your network in responding to pings and SNMP requests. The description directs the wizard to establish the length of timeouts.

To describe the reliability of your network, choose one of these options that correspond to the reliability of your network when responding to ping and SNMP requests.

Very reliable
This description states that the network must be reliable when responding to ping and SNMP requests. Select this option to allow the wizard to apply very short timeouts, without any retries. This option is appropriate for a very reliable network and results in fast discoveries. If you requested Fastest possible discovery time in the Discovery Optimization window, then the timeouts set by this option are even shorter.

Quite reliable
This description states that the network must be reliable for the most part when responding to ping and SNMP requests. Select this option to allow the wizard to apply slightly longer timeouts, with a single retry for both SNMP and ping requests.

Not very reliable
This description states that the network does not necessarily need to be reliable when responding to ping and SNMP requests. Select this option to allow the wizard to apply longer timeouts and two retries for SNMP requests and ping requests. Longer timeouts are suitable for less reliable networks.

Reviewing the configuration

On the Configuration Summary window, review your settings. You can also save the settings here, and, optionally, start the discovery with the settings that you configured.

To review your configuration settings, complete these steps.

1. Review the settings on the Configuration Summary window.

   Click any of the links to return to the relevant window to modify the settings as appropriate.
2. When you are satisfied with the discovery settings, select one of these options.
   - Select **Start Discovery**, to use the discovery configuration settings that you specified, and then click **Finish** to start the discovery.

- If you do not select **Start Discovery**, the discovery settings are saved when you click **Finish**.

Related tasks:

"Monitoring network discovery from the GUI" on page 131
From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

Discovering the network using the GUI

To perform a custom discovery, complete the tabs on the Network Discovery Configuration page. On these tabs, you can configure more complex discovery parameters than by using the Discovery Configuration Wizard.

Remember: To save configuration changes that you have made during a session, click the **Save** button before you log out, close the browser window, or close the Network Discovery Configuration tab. It is good practice to click **Save** as you move from tab to tab.

The parameters that you can set on the tabs of the Network Discovery Configuration are described in the topics that follow.

Most of the parameters that you can set on the Network Discovery Configuration page are optional.

For the discovery to run, at a minimum you must specify the following parameters:

- One seed device
- The correct SNMP community strings for the network to be discovered.

If any of the tabs contain data, this data is from earlier configurations. The data is held in the relevant discovery configuration file.

Scoping discovery

To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

You can define as many zones as necessary. You can add new zones, or edit or delete existing zones. Zones can be specified within zones: within a given inclusion zone, you can specify devices or subnets that are not to be detected. These devices are not pinged by the Ping finder or interrogated by the discovery agents. For example, you can define an include scope zone consisting of the Class B subnet 1.2.0.0/16, and within that zone you can specify an exclude scope zone consisting of the Class C subnet 1.2.3.0/24. Finally, within the exclude scope zone you could specify an include scope zone 1.2.3.128/26.

To scope the discovery:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **Scope**.
3. To add a new scope zone, click **New** . The Scope Properties page is displayed.
4. Complete the fields as follows and then click **OK**.

Scope By:

Select one of the following options:

Subnet

Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

You can specify a subnet or an individual IP address using these fields.

- For example, to specify a Class C subnet 10.30.2.0, type `10.30.2.0/24`, where `10.30.2.0` is the subnet prefix, and `24` is the subnet mask.
- To specify an individual device, type an IP address and a subnet mask of `32`. For example, type `10.30.1.20/32`.

Wildcard

Use an asterisk (*) as a wildcard.

For example, to specify a scope of all IP addresses that begin with the 10.30.200. subnet prefix, type `10.30.200.*`.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not support an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

Protocol

Select the required Internet protocol: IPv4 or IPv6.

Action

Define the subnet range as an inclusion zone or exclusion zone. If the subnet range is an inclusion zone that you intend to ping during the discovery, click **Add to Ping Seed List**. Clicking this option automatically adds the devices in the scope zone as a discovery seed devices.

Restriction: The `Add to Ping Seed List` option is not available for IPv6 scope zones. This prevents ping sweeping of IPv6 subnets, which can potentially contain billions of devices to be pinged. Ping sweeping of IPv6 subnets can therefore result in a non-terminating discovery.

5. To edit an existing scope zone, click the required row. On the Scope Properties page, edit the properties as described in step 4 on page 17.
6. To delete an existing scope zone, select the **Select** check box next to the required row or rows and click **Delete** ✖.
7. Click **Save** 💾.

If you are performing NAT address mapping, you must configure the NAT gateways and return to the **Scope** tab to set the address mapping.

Related concepts:

"Defining discovery zones to restrict discovery" on page 3
To restrict the discovery, you must define discovery zones. You can define discovery zones in several ways.

"Filters" on page 5
Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

"Scopes" on page 2
Define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude. The areas of the network to be included in the discovery process, or excluded from the discovery process are collectively known as the discovery scope.

"Types of scoping" on page 3
Network Manager offers several types of scoping.

Related tasks:

"Troubleshooting missing devices" on page 167
If a device that you expect to find in your network topology is not present, follow these steps to troubleshoot the problem.

"Configuring a multicast discovery" on page 34
Configure a multicast discovery by enabling the required agents and scoping the discovery.

Related reference:

"Main discovery stitchers" on page 341
This topic lists all discovery stitchers.

Appendix A, "Discovery databases," on page 183
There are various specialized databases that are used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.

"scratchTopology database schema" on page 263
The scratchTopology database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table name is: scratchTopology.entityByName.

"DiscoScope.cfg configuration file" on page 64
The DiscoScope.cfg configuration file can be used to configure the scope of a discovery.

"Quick reference for NAT discovery configuration" on page 118
Use this information as a step-by-step guide to configuring a NAT discovery..

Defining multiple inclusion zones

You can define multiple inclusion zones in the scope.zones table.

In the following example, three different IP inclusion zones are defined within a single insert.

```
insert into scope.zones
(
    m_Protocol,
    m_Action,
    m_Zones
)
values
(
        1,
        1,
        [
              {
                    m_Subnet="172.16.1.0",
                    m_NetMask=24
              },
              {
                    m_Subnet="172.16.2.*"
              },
              {
                    m_Subnet="172.16.3.0",
                    m_NetMask=255.255.255.0
              }
        ]
);
```

The above example defines three different IP inclusion zones each using a different syntax to define the subnet mask. Network Manager discovers:

- Any device that falls within the 172.16.1.0 subnet (with a subnet mask of 24, that is, 24 bits turned on and 8 bits turned off, which implies a netmask of 255.255.255.0).
- Any device with an IP address starting with "172.16.2", that is, in the 172.16.2.0 subnet with a mask of 255.255.255.0.
- Any device that falls within the 172.16.3.0 subnet with a mask of 255.255.255.0.

Related concepts:

"Defining discovery zones to restrict discovery" on page 3
To restrict the discovery, you must define discovery zones. You can define discovery zones in several ways.

Seeding discovery

To seed the discovery, provide the starting points from which to look for devices.

For the discovery to run, at a minimum you must specify the following parameters:

- One seed device
- The correct SNMP community strings for the network to be discovered.

Use the following methods to seed the discovery:

Ping finder
Seed the Ping finder with a device or subnet address at which the finder begins looking for devices. You can specify seeds for the Ping finder and save these seeds. You can separately decide whether to activate the Ping finder for the discovery.

File finder
Seed the File finder with a text file on the Network Manager host to which you have read access. This file must be a structured text file that contains the seeds in the form of IP addresses and device names in columns. You usually use a file that already exists on the Network Manager host. However, if you want to create a new file to hold the seeds, you need to have write permissions for the directory where you want to write the file.

There is also a mechanism to trigger a partial discovery based on receipt of a trap. This is performed by the Disco plug-in to the Event Gateway. For more information on the Disco plug-in, see the *IBM Tivoli Network Manager IP Edition Event Management Guide*.

When running an IPv6 discovery, ensure that the following conditions are met:

- There is at least one IPv6 seed device within each IPv6 scope.
- If you specify an IPv6 subnet as a seed, then ensure that the subnet is small by specifying a high value for the netmask.

By default, the Ping finder and File finder are switched on.

To seed the discovery:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **Seed**.
3. Optional: To switch off the Ping finder or File finder clear the **Use Ping Finder in Discovery** or **Use File Finder in Discovery** check boxes.
4. Add or edit a ping seed:
   - To add a new ping seed, click **New** 🗋.
   - To edit an existing ping seed, click the required entry in the list.

   The Ping Seed Properties page is displayed.
5. Complete the fields as follows and click **OK**.

   Seed by:
   Select one of the following options:

   IP Type an IP address.

   Subnet
   Specify a subnet and type the number of netmask bits. The **Netmask** field is automatically updated.

   Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not support an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

   Timeout
   Specify the time in milliseconds to wait for a reply from a pinged address before timing out.

   Retries
   Specify the number of times a device is to be repinged.
6. To delete an existing ping seed, select the **Select** check box next to the required row and click **Delete** ✖.
7. Add or edit a file seed:
   - To add a new file seed to the File finder, click **New** 🗋.
   - To edit an existing file seed, click the required entry in the list.

   The File Seed Properties page is displayed.
8. Complete the fields as follows and click **OK**.

   Filename
   Specify the path to the file on the host workstation that contains the seed data.

Delimiter

Specify the column delimiter. Use a regular expression if required. For example, if the Name and IP columns are separated by one or more tabs, then insert `[ tab_space ]+`, where *tab_space* is an actual tab character. To produce this tab character, create a tab in a text editor, copy the tab and paste it into the field.

Name Column

Type the column number of the column that contains the device names of the seed devices.

IP Column

Type the column number of the column that contains the IP addresses of the seed devices.

9. To delete an existing file seed, select the **Select** check box next to the required row and click **Delete** ✖.
10. Click **Save** 💾.

You can also seed a discovery by using the *Collector finder*. The collector finder retrieves topology data from an EMS. Topology data is collected by EMS collectors, which are software modules that retrieve topology data held in an EMS database, convert this data to XML format and pass this data to Network Manager IP Edition to stitch into the topology. You must seed the Collector finder to enable Network Manager IP Edition to find one or more EMS collectors.

Related reference:

"DiscoPingFinderSeeds.cfg configuration file" on page 61
The DiscoPingFinderSeeds.cfg configuration file is used for seeding the Ping finder and restricting device detection.

"DiscoCollectorFinderSeeds.cfg configuration file" on page 56
The `DiscoCollectorFinderSeeds.cfg` configuration file defines how topology data is acquired from Element Management System (EMS) collectors during discovery.

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

IPv6 subnet mask sizes

There are potentially billions of devices to be pinged within a single IPv6 subnet. To ensure that discovery completes, you must specify a sufficiently large netmask if you specify an IPv6 subnet as a ping seed.

The following table provides examples of IPv6 subnet mask sizes configured within ping seeds and the corresponding estimated time required to ping the devices in the subnet. The estimated times are based on spacing the pings by 100 ms between pings. This table shows that it is best to limit the size of IPv6 subnet masks in your subnet seeds.

Table 1. Ping response times for IPv6 subnet masks

IPv6 subnet mask size	Number of IPv6 addresses in subnet	Estimated ping time for subnet
120	256	26 seconds
112	65536	1 hour 48 minutes

Table 1. Ping response times for IPv6 subnet masks (continued)

IPv6 subnet mask size	Number of IPv6 addresses in subnet	Estimated ping time for subnet
100	268 million	Approximately 8.5 years

The time estimates shown in the table refer to time taken to ping all the seeds in a subnet seed specified for the Ping finder. It would take longer to complete the discovery as there will be many more devices to ping within the discovery scope.

Configuring device access

Specify SNMP community strings and Telnet access information to enable helpers and Network Manager polling to access devices on your network.

Note the following information about the SNMP helper and Telnet helper:

SNMP helper
: You must specify SNMP community strings for the SNMP helper and polling operations to access devices on your network. You might need to enter a community string more than once. For example, once for SNMPv1, once for SNMPv2, and once for SNMPv3.

Telnet helper
: Enter the relevant device prompts, login ID, and password for the Telnet helper and the discovery agents that use Telnet. You can configure Telnet-privileged access properties. The privileged access mode allows commands to be run that might change the configuration of the device. By default, when the discovery accesses the device using Telnet, access is granted in user mode. This mode allows the running of basic commands only, such as those commands that show the status of the system. This default access mode is a safety feature to prevent the discovery making any device configuration modifications without an explicit change to privileged mode.

Community strings and Telnet access data can be *global*, which means that the discovery tries the community string for every device it encounters, or restricted to specific subnets (that is, used only on devices within a specific subnet), or even restricted to specific devices. Specifying community strings and Telnet access data by subnet results in a more efficient and faster discovery. In general, the more specific the credentials, the faster the discovery will determine the correct credentials.

Note: Speed of discovery related to community string settings in the GUI only affects the initial discoveries. Once Network Manager has identified the correct community strings, it stores this information in the NCMONITOR relational database. Subsequent discoveries access this database for SNMP cmmunity strings and other SNMP-related device access information.

For the discovery to run, at a minimum you must specify the following parameters:

- One seed device
- The correct SNMP community strings for the network to be discovered.

You can also configure the SNMP Helper to use the GetBulk operation when SNMP v2 or v3 is used. Use of the GetBulk operation improves discovery speed. For more information, see the *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*.

When discovering devices using SNMPv3, the Cisco switches must have the VLAN context added to the view group for each VLAN.

To configure device access:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **Passwords**.
3. To add a new SNMP community string, click **New** . The SNMP Password Properties page is displayed.
4. Complete the fields as follows and then click **OK**:

   Community String

   > Type a name. When you save the community string, the name is encrypted, but on the GUI, the value is always displayed unencrypted. For speed of discovery, order the SNMP strings by frequency, with the most common strings first.
   >
   > **Restriction:** It is best practice not to use the at symbol (@) in community strings. Using this symbol in a community string can cause problems connecting to devices at discovery time.

   Apply to

   > The discovery completes more quickly if you specify the correct scope of the community strings. Select one of the following options:
   >
   > **All Devices**
   >
   > > Select this option if the community string is global.
   >
   > **IP Address**
   >
   > > Select this option if the community string is specific to an IP address, and type the IP address.
   >
   > **Subnet**
   >
   > > Select this option if the community string is specific to a subnet. Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

   SNMP Version

   > Specify the version of SNMP for this SNMP community. If you specify `SNMP V3`, complete the following additional fields:
   >
   > **Security Name**
   >
   > > Type a name.
   >
   > **Level** Specify the required level of authentication and privacy.
   >
   > > **NoAuthNoPriv,**
   > >
   > > > Select this option for SNMP communities that have no authentication or private key. In this case there is no need to specify any passwords.
   > >
   > > **AuthNoPriv**
   > >
   > > > Select this option for SNMP communities that have an

authentication key but no private key. Then specify a password in the **Auth Password** field.

AuthPriv

Select this option for SNMP communities that have both an authentication and a private key. Then specify passwords in the **Auth Password** and **Private Password** fields.

Auth Type

Specify the type of encryption for the authentication password.

Restriction: The MD5 encryption option is not available if you are running a FIPS 140–2 installation of Network Manager.

Priv Type

Specify the type of encryption for the privacy password.

Restriction: The DES encryption option is not available if you are running a FIPS 140–2 installation of Network Manager.

SNMP Port

Specify the required port.

Timeout

Specify the time in milliseconds to wait for a reply before timing out.

Retries

Specify how many times you want the SNMP helper and polling operations to attempt to access a device.

5. Click **Move Up** ▲ and **Move Down** ▼ to arrange the SNMP community strings. Put the most frequently used strings at the top of the list.
6. Click **Save**.
7. To add Telnet access information, click **New.** 🗋 The Telnet Password Properties page is displayed.
8. Complete the fields as follows:

Apply to

Select one of the following options:

All devices

Select this option if the data applies globally.

IP address

Select this option if the string is specific to a device, and type the IP address of the device.

Subnet

Select this option if the string is specific to a subnet. Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

Username prompt

Type the prompt that you want to be displayed at login. If you do not know the exact format of the prompt. use a regular expression.

Username

Type the user name.

Password prompt
: Type the prompt that you want to be displayed when the password is required at login. If you do not know the exact format of the prompt, use a regular expression.

Password
: Type the password.

Console prompt
: Type the prompt that is displayed when you log in. If you do not know the exact format of the prompt, use a regular expression.

Access port
: Specify the port on which the Telnet helper and discovery agents attempt to access devices.

Timeout
: Specify the time in milliseconds to wait for a reply before timing out.

Use SSH
: Select this option to configure the Telnet Helper to use the Secure Shell (SSH) program.

9. Optional: To configure Telnet-privileged access mode properties:
   a. Click **Advanced**. The Telnet Privileged Access Mode Properties page is displayed.
   b. Complete the fields as follows and then click **OK**:

      Command
      : Type the command required to enter Telnet-privileged access mode. This command is typically `enable`.

      Password Prompt
      : Type the prompt that you want to be displayed when the password is required at login. If you do not know the exact format of the prompt, use a regular expression.

      Password
      : Type the required password for privileged mode.

      Console Prompt
      : Type the prompt that is displayed when you log in. If you do not know the exact format of the prompt, use a regular expression.

      Commands requiring mode:
      : Specify the commands that you want to make accessible from privileged mode. To add new commands, click **New...** and type the command in the **Priv command** field. The following commands are required to run in `enable` mode:
        - **`show run`**
        - **`show mac-address-table`**
        - **`show ip nat translation`**

10. Click **OK**. Click **Save** 💾.

When you save the Telnet password settings, the following passwords are automatically encrypted:

- Telnet password
- Telnet privileged mode password (if specified)

When you save the password settings, the following passwords are automatically encrypted:

- SNMP community string
- SNMP authentication password
- SNMP private password

If required, change the SNMP and Telnet encryption settings. For example, you can change the encryption key file, or switch off encryption.

Related tasks:

"Enabling the StandardMPLSTE agent" on page 112
To discover MPLS TE tunnels, you must enable the StandardMPLSTE agent and add the relevant SNMP community strings.

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

"Connectivity at the layer 3 network layer" on page 316
There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

Activating agents

You must enable the appropriate agents for the discovery you want to perform. You can specify agents for a full discovery or for a partial discovery.

You can speed up the time taken for a partial discovery by selecting only those agents essential to discover the new or modified devices. You might want to run a partial discovery if you know that new devices have been added to the network or that engineers have been working on a device and have added or removed components of this device.

Note: The more agents you run, the more data is retrieved from the network, and the slower the discovery.

To activate agents:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click one of the following tabs, based on your requirements:

Tab	Description
Full Discovery Agents	Select agents from this tab to run a full discovery.
Partial Discovery Agents	Select agents from this tab to run a partial discovery. **Note:** The **Reset** button in the Partial Discovery Agents window sets the partial agents to match the settings defined in the Full Discovery Agents window.

   The Agents List is displayed, showing all available discovery agents for the selected discovery option.

3. Select the check boxes next to the required agents. For descriptions of the agents, select an agent name.

   To select all agents required for a layer 3 discovery, select the **Layer 3** checkbox. To select all agents required for a layers 2 and 3 discovery, select the **Full Layer 2 and Layer 3 Discovery** checkbox.
4. Click **Save** 💾. If you have selected a invalid combination of agents, or a combination that might result in an inefficient discovery, a warning is displayed.
5. If applicable, follow the steps displayed in the warning:
   - If you selected an agent that must be run in conjunction with another agent or agents, the warning indicates that the additional agents will be selected as applicable. Click **OK** to select the agents, or click **Cancel**.
   - If you selected an agent that cannot be run in conjunction with another agent or agents, the warning indicates that the redundant agents will be automatically deselected. Click **OK** to deselect the recommended agent or click **Cancel**.

Related tasks:

"Enabling collector discovery agents" on page 101
By default, the collector discovery agents are not enabled. You must enable these agents if you are running a discovery that includes collector-based discovery.

"Configuring MPLS agents" on page 105
As part of MPLS discovery configuration you must enable one or more MPLS agents. You can also resolve the problem of duplicate IP addresses in different Virtual Private Networks (VPNs) by configuring the AsAgent agent.

Related reference:

Appendix C, "Discovery agents," on page 303
Use this information to support the selection of discovery agents to run as part of your discovery.

Setting discovery filters

Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

A filter is made up of one or more filter conditions. Filter conditions are defined in Object Query Language (OQL). You can add the following types of filter:

Prediscovery filters
Prediscovery filters prevent discovered devices from being polled for connectivity information.

Post discovery filters
Post-discovery filters prevent discovered devices from being passed to MODEL.

Note: To ensure that alerts are not raised for interfaces that are excluded by the post discovery filter, you must set the `RaiseAlertsForUnknownInterfaces` variable. To this, perform the following steps:

1. Edit the `$NCHOME/etc/precision/NcPollerSchema.cfg` configuration file.
2. Add the following line to the file:

```
update config.properties set RaiseAlertsForUnknownInterfaces = 1;
```

The steps for adding, editing, and deleting filters are identical for both types.

To set the discovery filters:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **Filters**.
3. To use a filter in the discovery, select a filter from the **Available filters** list and click **Add**. The filter is added to the **Selected Prediscovery Filter** field or the **Selected Postdiscovery Filter** field, depending on the type of filter.
4. To delete a filter, select a filter from the **Available filters** list and click **Delete**.
5. To add a new filter, or edit an existing filter, click **Filter Library**. The Filter Library page is displayed.
6. Add or edit the filter as follows:

Action	Instructions
Add a new filter	Click **Add** and type the required name in the **Name** field.
Edit an existing filter	Select the required filter from the list.

7. On the **Basic tab**, build the filter conditions as follows:
   a. Select the required field and comparator.
   b. Type the value for comparison with the selected field. See "Sample filter" for an example.
   c. Click **Add New Row** ➕ or **Delete This Row** ✖ to add or remove rows.
   d. Select **All** to combine multiple conditions in an AND relationship, or **Any** combine the conditions in an OR relationship.
   e. Click **Save**.
8. Optional: On the **Advanced** tab, type the required SQL WHERE clauses. For multiple conditions, use an AND or an OR relationship as appropriate. Click **Save**.

   Note: The filter is actually based on standard OQL formatting, though the GUI refers to the SQL clause.
9. Click **Close** to close the Filter Library, then click **Save** to save your filter settings.

Sample filter

The following example shows a filter condition for a prediscovery filter:

```
m_ObjectId not like 1\.3\.6\.1\.4\.1\.2\.3\.1\.
```

For more information on OQL syntax, see the *IBM Tivoli Network Manager IP Edition Language Reference*.

Related concepts:

"Filters" on page 5
Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

Related tasks:

"Troubleshooting missing devices" on page 167
If a device that you expect to find in your network topology is not present, follow these steps to troubleshoot the problem.

Related reference:

"Main discovery stitchers" on page 341
This topic lists all discovery stitchers.

Appendix A, "Discovery databases," on page 183
There are various specialized databases that are used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.

"scratchTopology database schema" on page 263
The scratchTopology database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table name is: scratchTopology.entityByName.

Available filter values

Use this reference information to familiarize yourself with the permissible values when you set discovery filters on the Network Discovery Configuration page.

Prediscovery filter values

When constructing a prediscovery filter, you can filter based on any of the fields in the Details.returns table. These fields are as follows:

m_Name

m_UniqueAddress

m_Protocol

m_ObjectId

m_Description

m_HaveAccess

m_UpdAgent

m_AddressSpace

In addition, by using the **Advanced** tab, you can construct filter rows using any of the fields from within the m_ExtraInfo field.

Postdiscovery filter values

When constructing a post-discovery filter, you can filter based on any of the fields in the scratchTopology.entityByName table. These fields are as follows:

EntityName
The unique name of a network entity.

Address
A list that contains an address for the object for layers 1-7 of the OSI model.

Description
The sysDescr or other description.

EntityType
The type of the entity.

EntityOID
The class of the device.

Status
The status of the entity.

IsActive
Whether the entity is active.

Contains
The entities or other containers that are contained by this entity.

UpwardConnections
The entities that this entity is physically a part of.

RelatedTo
The devices an entity is connected to.

ExtraInfo
Miscellaneous extra information.

In addition, by using the **Advanced** tab, you can construct filter rows using any of the fields from within the ExtraInfo field.

Configuring Domain Name System

You can specify the methods that the Domain Name System (DNS) Helpers use to perform domain name lookups.

Helpers are specialized applications that retrieve information from and about network devices for the discovery agents.

Each of method that you specify uses one of the following three domain methods:

DNS Server
A server on the network that is dedicated to performing domain name resolution.

File The name of a file held on the Network Manager host that contains IP addresses and host names in lookup table format.

System
The local DNS system on the Network Manager host.

Tip: You can define as many methods as is necessary. You can change the order in which these methods are retrieved by the DNS Helper so that the most commonly accessed method is retrieved first. This enables a more effective use of resources during the discovery.

To configure DNSs:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click the **DNS** tab.
3. Add a new DNS helper, or edit an existing helper as follows:

- To add a new DNS helper, click **New** .
- To edit an existing helper, click the name of the required helper.

The DNS Service Properties page is displayed.

4. Complete the fields as follows and then click **OK**.

   Service Name
   Type the name of the method.

   Type Select one of the following options:

   DNS Server
   Type the IP address of the required DNS server. In the **Timeout** field, specify the number of seconds to wait for a response from the DNS Server before timing out.

   File Type the name of the file that contains the domain lookup information. Specify the order in which this information appears in the lookup table by selecting one of the radio buttons:

   - **Name then IP**
   - **IP then Name**

   System
   Choose this option to use the local DNS system on the Network Manager server.

   Domain Suffix
   Specify the suffix to append to each device name after the name is looked up. The specified domain suffix is only added if no domain suffix is present in the device name.

   Note: If you expect the discovery to return some or all devices names with domain suffixes already appended, then you can specify a list of expected domain suffixes. The domain suffix value specified in the **Domain Suffix** field is not appended to any device names returned by the discovery with these expected suffixes. To specify a list of expected domain suffixes, you must configure the DiscoDNSHelperSchema.cfg configuration file from the command line.

5. Repeat steps 3 on page 31 to 4 to add or edit the required methods.
6. In the **Move** column, click **Move Up** and **Move Down** to arrange the methods in the order of most frequently-expected use, with the most frequently-used methods at the top.
7. Click **Save** .

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

"DiscoDNSHelperSchema.cfg configuration file" on page 56
The DiscoDNSHelperSchema.cfg configuration file defines access to DNS, which enables the discovery to do domain name lookups, by configuring the DNS helper.

Configuring NAT translation

To configure NAT translation to discover NAT environments, map the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

After activating NAT, you must map the discovery scope zones to the NAT address spaces. You do this on the **Scope** tab.

To configure NAT gateways:

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **NAT**.
3. Add a new NAT gateway, or edit an existing gateway:
   - To add a new NAT gateway, click **New** .
   - To edit an existing NAT gateway, click the IP address in the required row.

   The NAT Gateway page is displayed.
4. Complete the fields as follows and click **OK**:

   IP Address
   Type the public IP address of the NAT gateway device.

   Address Space
   Type the address space identifier that you want to use for the associated NAT domain.
5. Click **Save** .
6. To activate NAT translation for the discovery, select **Enable Network Address Translation (NAT) Support**. Click **Save** and then map the discovery scope zones to the NAT address spaces:
   a. Click **Scope**.
   b. Click a scope zone to edit it. The Scope Properties page is displayed.
   c. In the **Address Space** field, enter the NAT address space and click **OK**. The **Address Space** field appears on the Scope Properties only after the **Enable Network Address Translation (NAT) Support** has been selected.
   d. Repeat the previous two steps for all the required scope zones.
   e. Click **Save** .

   NAT Address Spaces Dynamic Distinct view is created automatically if **Enable Network Address Translation (NAT) Support** is turned on. Once discovery is complete, use the Network Views to visualize the NAT Address Spaces network view.

Related tasks:

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

"Configuring NAT discoveries" on page 115
Configure a Network Address Translation (NAT) discovery to discover NAT environments, by mapping the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

Related reference:

"Quick reference for NAT discovery configuration" on page 118
Use this information as a step-by-step guide to configuring a NAT discovery..

Configuring a multicast discovery

Configure a multicast discovery by enabling the required agents and scoping the discovery.

Related concepts:

"Types of scoping" on page 3
Network Manager offers several types of scoping.

Related tasks:

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Related reference:

"scope.multicastSource table" on page 205
The scope.multicastSource table defines which IPM routes to discover. This is particularly useful if you have multiple IPM route sources, since you can scope multicast discovery by IPM route source to focus on the sources of interest.

"scope.multicastGroup table" on page 204
The scope.multicastGroup table defines which multicast groups to discover and which details to retrieve from these groups.

Enabling the multicast agents

To discover multicast groups, you must enable the appropriate agents and add the relevant SNMP community strings.

To enable the agents, complete the following steps.

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click the **Full Discovery Agents** tab. The Agents List is displayed, showing all available discovery agents for the selected discovery option.
3. Click **Full Layer 2 and Layer 3 Discovery** > **Multicast**.
4. Select the check box next to the agents you want to enable.
   a. Enable the StandardPIM agent to discover protocol-independent multicast groups compliant with the RFC2934 PIM MIB.
   b. Enable the StandardIPMRoute agent to discover IP multicasting networks compliant with the RFC2932 IPMRoute MIB.
   c. Enable the StandardIGMP agent to discover multicast groups running the Internet Group Membership Protocol (IGMP).
5. Click **Save** 💾.

6. Optional: If you want to rediscover multicast groups, also enable the appropriate agents for partial discoveries.
7. Ensure that the SNMP community strings are configured correctly to access the devices in the multicast groups.

Related reference:

"Multicast agents" on page 323
Multicast agents retrieve data from devices participating in multicast groups and routes.

Scoping a multicast discovery

Configure which multicast groups and sources to discover using the **Multicast** tab.

To configure a multicast discovery, complete the following steps.

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click **Multicast**.
3. In the **Multicast Groups** section, create a new multicast group or edit an existing group:
   - To create a new group to discover, click **New** .
   - To edit an existing group, click the group name.

   The Multicast Group Properties page is displayed.
4. Define the scope properties using the following fields:

   Group Name
   Specify a name for this multicast group.

   PIM Mode
   Select whether to include or exclude Protocol Independent Multicast (PIM) data from the discovery. By default, PIM data is included.

   IPM Route Mode
   Select whether to include or exclude Internet Protocol Multicast (IPM) group data from the discovery. By default, IPM Group data is included.

   IGMP Mode
   Select whether to include or exclude Internet Group Management Protocol (IGMP) data from the discovery. By default, IGMP data is included.

   Protocol
   Only IPv4 is supported.

   Specify which Group subnets to add to Multicast Groups
   Use the following fields and buttons to add and delete group subnets:

   Subnet
   Enter a subnet and netmask for a group subnet to add to the multicast groups.

   Add Click **Add** the add this group.

   Delete Select a group subnet from the adjacent list and click **Delete** to delete the selected group.

   Note: Reserved multicast addresses are excluded from the scope by default.
5. Click **OK**.

6. To delete one or more groups, select the groups you want to delete and click the **Delete** button . To select or deselect all groups, click the **Select All** or **Deselect All** button.
7. In the **Multicast Sources** section, create a new multicast source or edit an existing source.
   - To create a new source to discover, click **New** .
   - To edit an existing source, click the source name.

   The Multicast Source Properties page is displayed.
8. Define the source properties using the following fields:

   IPM Route Mode
   Select whether to include or exclude the group:
   - **Unknown - use default**
   - **Include source**
   - **Exclude source**

   Protocol
   Only IPv4 is supported.

   Specify which Group subnets to add to Multicast Sources
   Use the following fields and buttons to add and delete group subnets:

   Subnet
   Enter a subnet and netmask for a group subnet to add to the multicast sources.

   Add Click **Add** the add this group.

   Delete Select a group subnet from the adjacent list and click **Delete** to delete the selected group.

   Specify which Source subnets to add to Multicast Sources
   Use the following fields and buttons to add and delete group subnets:

   Subnet
   Enter a subnet and netmask for a sources subnet to add to the multicast sources.

   Add Click **Add** the add this group.

   Delete Select a source subnet from the adjacent list and click **Delete** to delete the selected soource.
9. Click **OK**.
10. To delete one or more groups, select the groups you want to delete and click the Delete button . To select or deselect all groups, click the Select All or Deselect All button.
11. Click **Save** .

Advanced discovery parameters

Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

Set the advanced parameters on the **Advanced** tab of the Network Discovery Configuration page. After you have set the advanced parameters, click **Save** 💾.

Attention: Modify the advanced settings only if you are an experienced Network Manager user. If you modify the advanced parameters and the discovery does not work as expected, click **Reset** to restore the default settings.

"Advanced Finder Configuration"
"Advanced Ping Finder Configuration"
"Advanced Discovery Configuration" on page 39
"Advanced Telnet Helper Configuration" on page 38
"Advanced SNMP Helper Configuration" on page 38
"Advanced DNS Helper Configuration" on page 39
"Advanced Discovery Configuration" on page 39

Advanced Finder Configuration

To set advanced parameters for the File finder, use the following field:

Concurrent File Finders
Specify the number of threads to be used by the File finder. Each thread can process a different seed file simultaneously. If you have many seed files and spare resources on the discovery server, more threads might result in a faster discovery. If you have only one seed file, increasing the number of threads has no effect.

Advanced Ping Finder Configuration

To set the advanced parameters for the Ping finder, use the following fields:

Concurrent Ping Finders
Specify the number of threads to be used by the Ping finder. Each thread processes one pingFinder.pingRules insert at a time. Increasing the number of threads does not speed up a single large ping sweep but might speed up feedback of many addresses. However, you must balance speed against the resources of your machine and the ability of the ping receiver to process the ping responses in a timely manner. If the number of threads is too high, the ping receiver will fall behind, resulting in false ping failures and a loss of device discovery.

Studies have shown that the default number of 10 threads is optimal for most situations. You can gradually increase the number of threads and monitor the number of ping failures and make a note of time savings. Depending on the resources available, at a certain point the benefits start to decrease as resources are overloaded.

Default Timeout
Specify the maximum time, in milliseconds, to wait for a reply from a pinged address. If you know that network latency is low, a reduced wait

time might result in a faster discovery. A value that is too low for your network might result in devices not being discovered.

Default Number of Retries
Specify the number of times a device is to be pinged again following a failed initial ping.

Inter-Ping Time
Specify the interval in milliseconds between ping attempts made against the devices contained in a list or subnet. If network traffic resulting from the discovery is not an issue, this value can be reduced.

Allow Broadcast Pinging
To enable broadcast address pinging, select this check box.

Allow Multicast Pinging
To enable multicast address pinging, select this check box.

Advanced Telnet Helper Configuration

To set advanced parameters for the Telnet helper, use the following fields:

Concurrent Telnet Helpers
Specify the number of threads to be used by the Telnet helper. If you have many devices from which you want to access data using Telnet or SSH then increasing this value might result in a faster discovery. Typical examples of such devices are Catalyst switches, MPLS devices, and NAT gateways. If you change this value, be sure that your system is configured to allow at least this number of concurrent Telnet sessions.

Default Timeout
Specify the maximum time, in milliseconds, to wait for access to a device.

Number of Retries
Specify the number of times to try to connect to the device following a failed initial connection attempt.

Tip: You can also configure some other advanced settings in the `DiscoTelnetHelperSchema.cfg` file.

Advanced SNMP Helper Configuration

To advanced parameters for the SNMP helper, use the following fields:

Concurrent SNMP Helpers
Specify the number of threads to be used by the helper. If you have many devices with SNMP access and spare resources on the discovery server, more threads might result in a faster discovery. If you change this value, make sure that your system is configured to allow at least this number of concurrent SNMP sessions. This value must be more than the number of threads used by the Details discovery agent.

Timeout
Specify the maximum time, milliseconds, to wait for access to a device.

Number of Retries
Specify the number of attempts to retrieve one or more SNMP variables from a device after a failed initial attempt.

GetNext Slowdown
Specify the delay, in milliseconds, between each SNMP GetNext request.

The **m_GetNextSlowDown** parameter is applied when the number of separate GetNext requests issued in order to retrieve a non-scalar SNMP variable exceeds the value of the **m_GetNextBoundary** parameter.

GetNext Boundary
Specify the minimum number of GetNext requests to be issued when a non-scalar SNMP variable is retrieved from a device. The **m_GetNextBoundary** parameter is applied before the delay specified by the **m_GetNextSlowDown** parameter is introduced.

Advanced DNS Helper Configuration

To set advanced parameters for the DNS helper, use the following fields:

Concurrent DNS Helpers
Specify the number of threads to be used by the helper. If you change this value, be sure that your system is configured to allow at least this number of concurrent DNS sessions.

Default Timeout
Specify the maximum time, in milliseconds, to wait for a response from a device.

Advanced Discovery Configuration

To specify advanced feedback control, ping verification, and further advanced discovery parameters, use the following fields:

Enable Feedback Control
Specify whether to enable feedback control. Feedback means that the data returned by agents is used by the discovery to find other devices. Examples of feedback data include the IP addresses of remote neighbors, and addresses in the subnet within which a local neighbor exists.

No Feedback
Feedback is switched off for all discoveries and rediscoveries. Only the devices specified to the finders are discovered. This option means that discoveries and rediscoveries complete in the quickest possible time. However, the resulting network topology is incomplete unless you specify all devices that you want to discover as seeds.

Tip: Switch feedback off if you want to discover only a list of certain devices. Specify the devices you want to discover as seeds.

Feedback
Feedback is switched on for full discoveries, full rediscoveries, and partial rediscoveries. This option provides a complete topology in all situations but takes the longest.

Feedback Only on Full
This setting is on by default. Feedback is switched on for full discoveries and full rediscoveries, but switched off for partial rediscoveries.

Enabling Ping Verification
Specify whether the discovery checks for pingable interfaces. If a device is not pingable, the device is not polled for alerts.

Don't Check Pingability

None of the discovered interfaces are checked for whether they can be pinged. Interfaces are polled regardless of whether they are pingable at discovery.

Check Pingability

After discovery, every discovered interface is checked for whether they can be pinged. The check is run against the details.returns table. Interfaces that have an entry in this table are pingable. Interfaces that do not have an entry in this table are not pingable. The pingable interfaces are marked to be polled.

Detect Best Setting

This setting is on by default. If feedback control has been enabled, after discovery, every discovered interface is checked for whether they can be pinged. The check is run against the details.returns table. Interfaces that have an entry in this table are pingable. Interfaces that do not have an entry in this table are not pingable. The pingable interfaces are marked to be polled.

Restriction: This option works only when you select one of the following options from the **Enable Feedback Control** list: `Feedback` or `Feedback only on Full`.

Enable 'Allow Virtual'

Specify how you want the discovery to handle virtual IP addresses: [1].

Don't Allow Virtual

Does not discover virtual IP addresses.

Allow Virtual

Discovers virtual IP addresses. This setting is on by default.

Allow if in scope.special

Discovers virtual IP addresses only if the address is defined in the scope.special table. This table defines management IP addresses.

Enable VLAN Modelling

Enable this setting to model VLANs in this discovery. If you enable VLAN modeling, then you can partition discovered topologies based on VLAN membership. Disabling VLAN modeling reduces discovery time.

Enable SysName Naming

Enable this setting to name devices using the value of the SNMP sysName variable as the main source of naming information. The sysName variable must be set and must be unique within the network. Enabling this setting has no impact on the discovery time, because the sysName variable is retrieved by the Details agent by default.

Enable Caching of Discovery Tables

Enable this setting to cache data during the discovery process in order to enable data recovery if the Discovery engine, ncp_disco, fails. A discovery

[1] Devices are typically discovered using IP addresses retrieved by the AssocAddress agent. If a device is discovered using an IP address that was not retrieved by the AssocAddress agent, then the IP address is probably non-standard. This type of IP address is called a *virtual IP address*. Examples of virtual IP addresses are HSRP and VRRP addresses, which are shared by multiple devices for fault tolerance. Other examples include certain management interfaces that might be on a single device but do not appear in the IP table for security reasons. Virtual IP addresses include management addresses. A management address is an IP address whose only role is to manage the device. Management addresses are often on a separate network isolated from the customer traffic. These addresses are defined in the scope.special table.

running in this mode is slower than a standard discovery, because of the extra time required to store data on the disk throughout the discovery process.

Enable File Finder Verification

Enable this setting to use the Ping finder to verify the existence of devices specified in the files used by the File finder. If you enable this setting, ensure that the Ping finder is enabled. Enable this setting if you are not sure that the devices are still connected to the network. For example, you might want to enable this setting if your network is rapidly changing.

Enable Inference of Dumb Hubs

Configure the discovery to infer the existence of an unmanaged hub if the discovery finds a port that is connected to more than one item. The discovery then connects that port to an unmanaged hub, and also connects all the other port involved in the connection to the unmanaged hub. This type of discovery makes the topology much clearer. [2] By selecting this option, you instruct the discovery to activate the AddUnmanagedHub stitcher, which manages the inference of dumb hubs.

Enable Rediscovery Rebuild Layers

Enable this setting to rebuild the topology layers following a partial rediscovery. If you specify that topology layers are rebuilt following partial rediscovery, the result is an accurate topology showing all connectivity. However, the process of adding new devices takes longer.

Tip: To configure a partial rediscovery to run as quickly as possible, disable this option.

Enable RT-Based MPLS VPN Discovery

This setting is relevant to MPLS discoveries. Enable this setting to display provider edge devices only (RT-based MPLS discovery).

Enable Rediscovery of Related Devices

By default the remote neighbors of a device are not discovered, even if rediscovery of that device indicates that the remote neighbors have changed. The remote neighbors can be rediscovered on the next full rediscovery. Enable this setting if you want to change this default behavior and rediscover any changed remote neighbors when rediscovering that device.

Tip: To configure a partial rediscovery to run as quickly as possible, disable this option.

Enable ifName/ifDescr Interface Naming

Changes the default naming convention for discovered interfaces. Names interfaces using data from the SNMP interfaces table ifName and ifDescr fields as appropriate. For example, `Fa0/0`, `Gi 1.0.2:0`, `Gigabit Ethernet 4/1`. If you change the default naming convention for discovered interfaces, you must change the BuildInterfaceName stitcher to specify your naming convention.

[2] For example, if port number 1 on switches A, B, C, and D are all connected to each other, then this discovery option inserts an unmanaged hub E into the topology. The discovery then connects port 1 on switch A to E, rather than to B. It similarly connects port 1 on switches B, C, and D to E. A connection is created between device A and hub device E. Hub device E, and the connections to this device, might not actually exist in reality.

Tip: Some devices might report interface names and descriptions that are too long to display properly in the topology display. If there are devices that report long or incorrect interface names and descriptions, disable this setting.

Enable Inference of PEs using BGP data on CEs

Discovers intervening provider networks as a “third-party” object on multiple networks that run across a provider network. Examples of this type of network include enterprise VPNs across a provider MPLS core network. Select this option if you want to link all your networks in a single topology and perform root cause analysis (RCA) across your networks.

This option infers the existence of inaccessible provider-edge (PE) devices by using the BGP data on the customer-edge (CE) devices that point to the PE devices. In order to discover this BGP data, the BGP discovery agents must be enabled.

You can also optionally specify which of the inferred PE devices are valid devices, by populating the scope.inferMPLSPEs table, using standard format scope entries, as in the scope.zones table. If populated this table allows you to define which IP addresses you see on CE devices that you consider valid PE devices. Use this option when you have inaccessible devices that are connected by BGP but which are not actually PE devices.

Enable Inference of MPLS CE routers on /30 subnets

Generates Service-Affected Events on customer VPNs. Select this option if you are a service provider without access to customer CE routers.

Related concepts:

"About Service Affected Events" on page 104
A Service Affected Event (SAE) alert warns operators that a critical customer service has been affected by one or more network events.

"Option to rebuild topology layers" on page 301
You can specify whether to rebuild the topology layers following a partial rediscovery. Using this option, you can increase the speed of partial rediscovery.

Related tasks:

"Configuring MPLS discovery method" on page 108
You can configure MPLS discovery in either of two ways: Route Target (RT)-based discovery; Label Switched Path (LSP)-based discovery.

"Inferring the existence of CE routers" on page 109
You can infer the existence of your customers' CE routers by making specifications in the advanced discovery configuration options within the Discovery Configuration GUI.

Related reference:

"Main discovery stitchers" on page 341
This topic lists all discovery stitchers.

"Failover database" on page 270
Failover recovery with the failover database is not to be confused with agent and finder failover recovery, which are configured directly from the disco.config table. When selected, agent and finder failover recovery operate regardless of whether recovery with the failover database is implemented.

"disco.config table" on page 183
The config table configures the general operation of the discovery process.

"inferMPLSPEs table" on page 202
Use the inferMPLSPEs table when enabling inference of inaccessible provider-edge (PE) devices by using the BGP data on the customer-edge (CE) devices. This table enables you to optionally specify which zones to process to determine which of the inferred PE devices are valid devices.

Starting a discovery

After you configure a discovery, you can start and, if necessary, stop the discovery.

Make any required discovery configuration changes before you launch the discovery.

You can start the following types of discovery:

Discovery
Run a full discovery to discover your network for the first time, or to refresh the network topology if you know the network has changed.

Partial discovery
Run a partial discovery if you know that the changes to your network are limited to a small number of devices. You need to configure scoping and seeding as part of starting each partial discovery. If the relationship of the devices that are in scope with their neighboring devices has changed, then the neighboring devices may also be discovered. If the partial discovery needs to discover a large amount of devices based on connectivity information, then a full discovery is started.

Note: If you stop a running discovery, you must then do a full discovery before you are able to do a partial discovery.

To start a discovery, complete the following steps.

1. Click **Discovery** > **Network Discovery Status**.
2. Select the domain in which you want to run a discovery from the **Domain** menu. You can start to type the name of the domain, and matching domains are listed below the **Domain** field.
3. Start a full or partial discovery:
   - To start a full discovery, click **Start Discovery** ▶ only. The discovery starts.

     Important: In Network Manager V3.9 there is no longer any need to press **Stop Discovery** and then **Start Discovery** in order to pick up discovery configuration changes. Network Manager picks up any saved discovery configuration changes when you click **Start Discovery** ▶ .
   - To start a partial discovery, click the downward-facing arrow next to the **Start Discovery** button ▶ ▾ and select **Start Partial Discovery** from the menu (if a full discovery has not been run since the last time that the discovery engine, `ncp_disco`, was started, the option to start a partial discovery is grayed out). The Partial Discovery window is displayed. Specify the IP addresses and subnets that contain the devices to be discovered:

   a. Under **Partial Discovery**, select the required nodes and subnets.

   b. To add a new subnet or node, click **New.**

   c. Complete the fields as follows and click **OK**:

      Rediscover
      Select one of the following options:

      IP Address
      Type the required IP address.

      Subnet
      Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

   d. To add new scope zones, click **Scope**.

   e. To add a new discovery scope zone click **New** 🗋 . To edit an existing scope zone, click the required entry in the list.

   f. Complete the fields as follows and click **OK**:

      Scope By:
      Select one of the following options:

      Subnet
      Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

      You can specify a subnet or an individual IP address using these fields.

      - For example, to specify a Class C subnet 10.30.2.0, type `10.30.2.0/24`, where `10.30.2.0` is the subnet prefix, and `24` is the subnet mask.

- To specify an individual device, type an IP address and a subnet mask of 32. For example, type `10.30.1.20/32`.

Wildcard

Use an asterisk (*) as a wildcard.

For example, to specify a scope of all IP addresses that begin with the 10.30.200. subnet prefix, type `10.30.200.*`.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not support an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

Protocol

Select the required Internet protocol: IPv4 or IPv6.

Action

Define the subnet range as an inclusion zone or exclusion zone. If the subnet range is an inclusion zone that you intend to ping during the discovery, click **Add to Ping Seed List**. Clicking this option automatically adds the devices in the scope zone as a discovery seed devices.

Restriction: The `Add to Ping Seed List` option is not available for IPv6 scope zones. This prevents ping sweeping of IPv6 subnets, which can potentially contain billions of devices to be pinged. Ping sweeping of IPv6 subnets can therefore result in a non-terminating discovery.

g. Click **OK** then click **Go**. When a full or partial discovery is running, the **Start Discovery** button is toggled off ▷.

4. To stop a discovery, click **Stop Discovery** 🛑. The discovery might take a short time to stop, during which time both the **Start Discovery** and **Stop Discovery** buttons are toggled off. If you stop a discovery, you cannot then do a partial discovery until after the next full discovery.

   Note: When you stop a discovery, the discovery cache is lost. This is why you must wait for the completion of the next full discovery before being able to perform a partial discovery. It is possible to configure the Discovery engine to save the discovery cache as the discovery is running, which would enable you to run a partial discovery immediately following the manual stop of a discovery. You can configure the Discovery engine to save the discovery cache by clicking **Enabling Caching of Discovery Tables** in the **Advanced** tab.

While the discovery is running, you can monitor the progress of the discovery.

After the discovery is complete, the **Start Discovery** button is toggled on, and you can run another full or partial discovery at any time. If the Event Gateway Disco plug-in is enabled, then a new discovery can be triggered automatically when a reboot event (event ID of `NmosSnmpReboot` triggered by the rebootDetection poll policy) is received.

Related concepts:

"About types of discovery" on page 1
Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Related tasks:

"Monitoring network discovery from the GUI" on page 131
From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

"Starting partial discovery from the GUI" on page 158
Starting a partial discovery involves defining a seed and scopes.

"Troubleshooting an idle discovery" on page 168
If you start the discovery, and after some minutes no devices have been discovered, follow these troubleshooting steps.

Schemas and tables for GUI discovery parameters

Use this reference information to learn to which schemas and table the settings made on the tabs of the Network Discovery Configuration page are saved.

The following table describes the tables to which the settings made on each tab of the Network Discovery Configurationpage are saved. In these tables, *DOMAIN_NAME* represents the name of the network domains in your deployment, for example NCOMS.

Table 2. Schemas and tables to which the discovery parameters are mapped

Network Discovery Configuration tab	Description	Schema or table name
Scope	The zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.	`DiscoScope.`*`DOMAIN_NAME`*`.cfg`
Seed	The location from which to begin discovering devices. This might be one or more IP addresses, or subnet addresses. To seed the discovery, the following *finders* are used: Ping finder and File finder.	Ping finder: `DiscoPingFinderSeeds.`*`DOMAIN_NAME`*`.cfg` File finder: `DiscoFileFinderParseRules.`*`DOMAIN_NAME`*`.cfg`
Full Discovery Agents and **Partial Rediscovery Agents**	The discovery agents to be used to investigate device connectivity. Default agents are provided for the type of discovery you want to perform, for example a layer 2 or layer 3 discovery. You can select different set of agents for full discoveries and for partial discoveries. The agents vary because connectivity information varies with the technology of the hardware in the network.	`DiscoAgents.`*`DOMAIN_NAME`*`.cfg`
Device Access	SNMP community strings and Telnet parameters that Network Manager uses to interrogate devices that use SNMP and Telnet.	SNMP community strings: `SnmpStackSecurityInfo.cfg` Telnet access: `TelnetStackPasswords.cfg`

Table 2. Schemas and tables to which the discovery parameters are mapped (continued)

Network Discovery Configuration tab	Description	Schema or table name
Filters	Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Prediscovery filters prevent discovered devices from being polled for connectivity information. Post-discovery filters prevent discovered devices from being passed to MODEL.	`DiscoSchema.`*`DOMAIN_NAME`*`.cfg`
DNS	Access to DNS services that are used to perform domain name lookups.	`DiscoDNSHelperSchema.cfg`
NAT	The data that provides the discovery mappings between address space data and real device IP addresses to facilitate further discovery.	`DiscoSchema.`*`DOMAIN_NAME`*`.cfg`
Multicast	Multicast groups and sources used by the Discovery engine to configure multicast scopes.	`DiscoScope.`*`DOMAIN_NAME`*`.cfg`
Advanced	Advanced settings control features of the discovery such as concurrent processes and time-outs. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.	`DiscoSchema.`*`DOMAIN_NAME`*`.cfg`

Discovering the network using the command-line interface

As an experienced user, you can configure and track a network discovery using configuration files and database queries.

Related tasks:

"Monitoring discovery from the command line." on page 136
When the **`ncp_disco`** process is running, you can monitor the progress of the discovery by using the OQL Service Provider, the **`ncp_oql`** process, to query the discovery databases to determine what is happening at any time.

Discovery configuration files

Experienced users can configure the discovery by editing the discovery configuration files.

To configure the discovery using the command-line interface (command line), edit the discovery configuration files and create or edit inserts into the databases of the discovery processes.

Note: The DiscoSchema.cfg configuration file contains the schemas for all the discovery databases. Unlike the files listed below, the DiscoSchema.cfg configuration file contains no insert statements. You can view this file but it must not be edited.

When ncp_disco is running, it periodically scans the agents and stitchers directories and loads any new or modified stitcher and discovery agent definitions.

Table 3 shows which configuration files to edit to configure the discovery, and whether the configuration can also be done using the Discovery Configuration GUI.

Table 3. User-editable discovery configuration files

Discovery configuration task	Configuration file	GUI tab
Scoping discovery		
Defining inclusion and exclusion zones	DiscoScope.cfg	**Scope**
Ignoring discovery scope	DiscoScope.cfg	**Scope**
Seeding discovery		
Seeding	DiscoPingFinderSeeds.cfg	**Seed**
Running multiple instances of a finder		
Configuring broadcast and multicast address pinging	DiscoPingFinderSeeds.cfg	**Advanced**
Using the File finder	DiscoFileFinderParseRules.cfg	**Seed**
Enabling File finder device verification	DiscoConfig.cfg	**Advanced**
Enabling Ping verification	DiscoConfig.cfg	
Using and configuring the Collector finder	DiscoCollectorFinderSeeds.cfg	
SNMP		
Configuring SNMP community strings and passwords	SnmpStackSecurityInfo.cfg	**Passwords**
Configuring the SNMP helper	DiscoSnmpHelperSchema.cfg	**Advanced**
Overriding SNMP helper settings for specific devices and subnets		
Telnet		
Configuring Telnet access to network devices	TelnetStackPasswords.cfg	**Passwords**
Configuring the Telnet helper	DiscoTelnetHelperSchema.cfg	**Advanced**
Configuring a context-sensitive discovery	DiscoConfig.cfg	
Agents		
Enabling and disabling discovery agents	DiscoAgents.cfg	**Full Discovery Agents** **Partial Rediscovery Agents**

Table 3. User-editable discovery configuration files (continued)

Discovery configuration task	Configuration file	GUI tab
Filtering devices sent to the agents	Discovery agent definition files	**Filters**
Filtering topology data returned by an agent	Discovery agent definition files	
Filtering topology data returned by all agents	DiscoAgentReturns.filter	
Changing the number of threads used by an agent	DiscoAgents.cfg	
Enabling multi-threaded operation for Perl agents	Discovery agent definition files	
Enabling and disabling partial matching	IpForwardingTable.agnt agent definition file (for modern devices that use RFC2096) IpRoutingTable.agnt agent definition file (for older devices that use RFC1213).	
Restricting discovery		
Restricting device detection	DiscoScope.cfg DiscoPingFinderSeeds.cfg	**Scope** **Seed**
Restricting device interrogation	DiscoScope.cfg	
Restricting device instantiation		
Configuring the DNS helper services	DiscoDNSHelperSchema.cfg	**DNS**
Configuring a NAT discovery	NATTextFileAgent agent NATGateway agent	**NAT**
Setting advanced configuration		
Advanced File Finder Configuration Advanced Ping Finder Configuration Advanced DNS Helper Configuration Advanced SNMP Helper Configuration Advanced Telnet Helper Configuration	DiscoFileFinderParseRules.cfg DiscoPingFinderSeeds.cfg DiscoDNSHelperSchema.cfg DiscoSnmpHelperSchema.cfg DiscoTelnetHelperSchema.cfg **Note:** As an experienced user, you can set more advanced configuration parameters in the configuration files than are available in the Advanced tab of the GUI.	**Advanced**

Discovery agent definition files

The discovery agent definition files define the operation of the discovery agents.

Filtering devices using the definition files

Note: Network Manager kills all discovery agents at the end of data collection stage 3. This ensures that the next discovery restarts the agents and forces the agents to reread their configuration files at the beginning of a discovery, thereby detecting any changes to the configuration files.

You can apply a filter to a discovery agent by editing the supported devices filter within the DiscoAgentSupportedDevices( ); section of the discovery agent definition file ($NCHOME/precision/disco/agents/*.agnt). All discovery agents have a definition file in this directory, regardless of whether the agent is text-based or precompiled.

The supported devices filter is a filter against the attributes of the agentTemplate.despatch table.

The DiscoAgentSupportedDevices( ); section accepts full OQL comparison tests using comparison operators such as `like`, `<` , `>` , `=` , and <>. Detailed information about comparison tests in OQL can be found in the *IBM Tivoli Network Manager IP Edition Language Reference*.

Tip: Altering agent definition files can introduce parse errors. To check your agent for parse errors, run the agent in debug mode and examine the debug output.

Example: discovering devices that use CDP

The CDP discovery agent, defined in the $NCHOME/precision/disco/agents/CDP.agnt agent file, must be enabled before the discovery to detect devices that use CDP. Enable the CDP agent by setting the value of the m_Valid column to 1, as shown in the following insert.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
)
values
(
        'CDP', 1, 7, 0, 3
);
```

Sample: filtering devices sent to the CDP agent

The following example shows the DiscoAgentSupportedDevices(); section of the `CDP.agnt` agent definition file. Only network entities that match the specified Object IDs are processed by the CDP agent, that is, only devices that use the Cisco Discovery Protocol. The CDP agent does not process devices with the Object ID 1.3.6.1.4.1.9.1.226.

```
            DiscoAgentSupportedDevices
            (
                        " (
                                    ( m_ObjectId like '1\.3\.6\.1\.4\.1\.9\..*' )
                                    AND
                                    ( m_ObjectId <> '1.3.6.1.4.1.9.1.226' )
                        ) "
            );
```

Sample: using wildcards in device filters

The following example shows the use of wild cards in the IP address column. The agent only accepts devices with an IP address beginning 10.10.2.

```
DiscoAgentSupportedDevices
(
            " ( m_UniqueAddress like '10\.10\.2\..*' ) "
);
```

Example: using multiple device filter conditions

The following example shows the combination of multiple filter conditions. The agent accepts only devices that have the Object ID 1.3.6.1.4.1.9.5.7.. have an IP address starting with 10.10.. and do not have the name clandestine.

```
DiscoAgentSupportedDevices
(
            "(
                    ( m_ObjectId = '1.3.6.1.4.1.9.5.7' )
                    AND
                    ( m_UniqueAddress like '^10\.10\..*' )
                    AND
                    ( m_Name not like '.*[cC]landestin[eE].*' )
            )"
);
```

Enabling multi-threaded operation for Perl discovery agents

The number of threads used by discovery agents is set in the DiscoAgents.cfg configuration file. Perl agents must have multi-threaded operation enabled before the setting in the DiscoAgents.cfg configuration file has any effect.

To enable multi-threaded operation for a Perl discovery agent, add the following line to its definition file:

```
DiscoAgentDefaultThreads( 10 );
```

The insert above specifies that the agent uses 10 threads by default. If you set a different number of threads in the DiscoAgents.cfg configuration file, that value overrides the value in the agent definition file.

Restriction: Many of the add-on CPAN modules often used with Perl are not thread safe. Perl discovery agents using such modules might need to be restricted to a single thread.

Filtering topology data returned by a discovery agent

To filter topology data returned by a single agent, define a filter within the relevant agent (.agnt) file.

Sample: filtering out subscriber cable-modem interfaces

The CMTS.agnt agent file retrieves data from cable modems connected to a cable modem terminating services device. This example describes a filter added to the CMTS.agnt file which filters out subscriber cable modem interfaces from topology data returned for the CMTS devices. The example filter is as follows:

```
DiscoAgentReturnsFilterList
{
      DiscoReturnsFilter
      {
            "(
             m_LocalNbr->m_IfType = 229
            )"
      }
};
```

Sample: defining multiple topology filters

The following example illustrates how to define multiple topology data filters within an agent. The first filter specifies that each time a record is returned where the interface ifIndex value is 4, then the m_Name, m_HaveAccess, m_LocalNbr->m_SubnetMask, and m_RemoteNbr->m_RemoteNbrPhysAddr fields must be deleted from the record. The second filter deletes records returned when the interface ifIndex value is 5.

```
DiscoAgentReturnsFilterList
{
      DiscoReturnsFilter
      {
            "(
             m_LocalNbr->m_IfIndex = 4
            )"
            DiscoDeleteFields {
                  "m_Name",
                  "m_HaveAccess",
                  "m_LocalNbr->m_SubnetMask",
                  "m_RemoteNbr->m_RemoteNbrPhysAddr",
            }
      }
      DiscoReturnsFilter
      {
            "(
             m_LocalNbr->m_IfIndex = 5
            )"
      }
};
```

Sample: Disabling partial matching

The following example could be appended to the IpForwardingTable.agnt definition file to ensure that if a router with m_ObjectId='1.3.6.1.4.1.9.1.48' is discovered (that is, a Cisco 7505 router), partial matching is attempted only when the router is running IOS version 12.2 or higher.

```
DiscoRouterPartialMatchRestrictions
(
            "(m_ObjectId='1.3.6.1.4.1.9.1.48', m_OSVersion>='12.2',
            m_MibVar='sysDescr')"
);
```

Sample: Disabling partial matching using wildcards

The following example ensures that partial matching is not used on Cisco 2600 routers, Cisco 7505 routers running an IOS revision lower than 12.2, and Redstone routers.

```
DiscoRouterPartialMatchRestrictions
(
            "(m_ObjectId='1.3.6.1.4.1.9.1.209'),
            (m_ObjectId='1.3.6.1.4.1.9.1.48', m_OSVersion>='12.2',
             m_MibVar='sysDescr'),
            (m_ObjectId like '1\.3\.6\.1\.4\.1\.2773\..*')"
);
```

Related reference:

"disco.agents table" on page 193
The agents table specifies the discovery agents that DISCO uses for the discovery. Every agent that you want to run must have an insertion into the disco.agents table within the DiscoAgents.cfg configuration file that enables that agent (set m_Valid=1). If m_Valid=0, the agent is not run.

"Agent Template database" on page 274
The databases of each discovery agent are based on a template called the agentTemplate database.

DiscoAgents.cfg configuration file

The DiscoAgents.cfg configuration file defines which agents run during a discovery.

Database table used

The DiscoAgents.cfg configuration file can be used to configure inserts into the disco.agents database table.

Sample: enabling the IpRoutingTable discovery agent

The following example activates the IpRoutingTable discovery agent.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
    )
values
(
        'IpRoutingTable', 1, 0, 0, 2
    );
```

Sample: enabling the Details and Associated Address agents

The following example OQL inserts activate the Details and Associated Address agents.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
    )
values
(
        'Details', 1, 0, 0, 1
    );

insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
    )
values
(
        'AssocAddress', 1, 0, 0, 2
    );
```

Sample: enabling the ARP cache agent

The ARP Cache agent assists in MAC address-to-IP address resolution during the discovery. You must enable this agent to run during a layer 2 discovery. The following example shows how to ensure that the ARP Cache agent runs during a discovery.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
)
values
(
        'ArpCache', 1, 0, 0, 2
);
```

Sample: deactivating the StandardSwitch and SuperStack3ComSwitch agents

The following example deactivates the StandardSwitch and the SuperStack3ComSwitch discovery agents.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
    )
values
(
        'StandardSwitch', 0, 1, 1, 3
    );

insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
    )
values
(
        'SuperStack3ComSwitch', 0, 1, 1, 3
    );
```

Sample: changing the number of threads used by the IpRoutingTable discovery agent

The following example sets the number of threads used by the IpRoutingTable discovery agent to 50. Increasing the number of threads used by an agent allows the agent to process more devices at once, and can speed up discovery. However, increasing the number of threads used by an agent also uses more memory.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence, m_NumThreads
    )
values
(
        'IpRoutingTable', 1, 0, 0, 2, 50
    );
```

Sample: changing the number of threads used by the NMAPScan Perl discovery agent

The following example sets the number of threads used by the NMAPScan Perl discovery agent to 50. To define the number of threads used by a Perl discovery agent, you must first enable multiple threads for that agent in the discovery agent definition file.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence, m_NumThreads
    )
```

```
values
(
        'NMAPScan', 1, 0, 0, 2, 50
    );
```

Related reference:

"disco.agents table" on page 193
The agents table specifies the discovery agents that DISCO uses for the discovery. Every agent that you want to run must have an insertion into the disco.agents table within the DiscoAgents.cfg configuration file that enables that agent (set m_Valid=1). If m_Valid=0, the agent is not run.

DiscoAgentReturns.filter configuration file

The DiscoAgentReturns.filter configuration file allows you to apply a topology data filter to data returned by all discovery agents.

Filtering topology data returned by all agents

The $NCHOME/precision/disco/agents/DiscoAgentReturns.filter configuration file filters the same topology data from all of the agent returns tables. The syntax used in this file is the same as the syntax used in topology filters in the discovery agent definition files.

Sample: filtering out subscriber cable-modem interfaces

The following example filters out subscriber cable modem interfaces from topology data:

```
DiscoAgentReturnsFilterList
{
      DiscoReturnsFilter
      {
            "(
             m_LocalNbr->m_IfType = 229
            )"
      }
};
```

Related concepts:

"Agents" on page 303
Discovery agents retrieve information about devices in the network. They also report on new devices by finding new connections when investigating device connectivity. Discovery agents are used for specialized tasks. For example, the ARP Cache discovery agent populates the Helper Server database with IP address-to-MAC address mappings.

DiscoARPHelperSchema.cfg configuration file

The DiscoARPHelperSchema.cfg configuration file performs IP address to MAC address resolution.

Database used

The DiscoARPHelperSchema.cfg configuration file defines inserts into the ARPHelper.configuration database table.

Sample: Configuring the ARP helper

The following example insert configures the ARP helper to use one thread.

```
insert into ARPHelper.configuration
(
        m_NumThreads
)
values
(
        1
);
```

Related reference:

"The ARP helper database" on page 248
The ARP helper database is defined by the DiscoARPHelperSchema.cfg configuration file Its fully qualified database table name is ARPHelper.configuration.

DiscoCollectorFinderSeeds.cfg configuration file

The `DiscoCollectorFinderSeeds.cfg` configuration file defines how topology data is acquired from Element Management System (EMS) collectors during discovery.

Database used

The `DiscoCollectorFinderSeeds.cfg` configuration file defines inserts into the collectorFinder database.

Note that there is another file associated with the collectorFinder database, the DiscoCollectorFinderSchema.cfg file, but you should not need to alter this file.

Sample: configuring a single collector

The following example seeds a single collector running on the local server. The example does not specify values for other fields, such as m_DataSourceId and m_NumRetries, and they automatically take the default values from the configuration table.

```
insert into collectorFinder.collectorRules
        ( m_Port)
values
        ( 8082 );
```

Related reference:

"collectorFinder database" on page 226
The collectorFinder database defines the operation of the Collector finders.

DiscoDNSHelperSchema.cfg configuration file

The DiscoDNSHelperSchema.cfg configuration file defines access to DNS, which enables the discovery to do domain name lookups, by configuring the DNS helper.

Database tables used

The DiscoDNSHelperSchema.cfg configuration file can be used to configure inserts into the following database tables:

- DNSHelper.configuration
- DNShelper.methods

Sample: configuring the DNS helper

The following example inserts configure the DNS helper using the information in the DNSHelper.configuration database table and the DNShelper.methods database table. The example shows inserts into the DNShelper.methods database table corresponding to the following method types:

- 0 - System
- 1 - DNS using `m_NameDomain` to specify a domain suffix to append to all discovered device names.
- 1 - DNS using `m_NameDomainList` to specify a list of expected domain suffixes.
- 2 - File

```
insert into DNSHelper.configuration
(
     m_NumThreads, m_MethodList, m_TimeOut
)
values
(
     1, ['HostsFile'] , 5
);

insert into DNSHelper.methods
(
     m_MethodName, m_MethodType
)
values
(
     "HostService", 0
);

insert into DNSHelper.methods
(
     m_MethodName, m_MethodType, m_NameServerAddr, m_TimeOut, m_NameDomain
)
values
(
     "abcIPv6DNS", 1, "2222:15f8:106:203:250:4ff:fee8:6d75", 3,
"tivlab.raleigh.ibm.com"
);

insert into DNSHelper.methods
(
     m_MethodName, m_MethodType, m_TimeOut, m_NameServerAddr, m_NameDomainList
)
values
(
     "defIPv6DNS", 1, 3, "2222:15f8:106:203:250:4ff:fee8:6d75",
     ['uk.eu.org',
     'fra.eu.org',
     'de.eu.org',
     'it.eu.org',
     'sp.eu.org']
);

insert into DNSHelper.methods
(
     m_MethodName, m_MethodType, m_FileName, m_FileOrder
)
values
(
     'HostsFile', 2, 'etc/hosts', 1
);
```

Related reference:

"The DNS helper database" on page 248
The DNS helper database is defined by the DiscoDNSHelperSchema.cfg configuration file. Its fully qualified database table names are: DNSHelper.configuration; DNShelper.methods.

DiscoFileFinderParseRules.cfg configuration file

The DiscoFileFinderParseRules.cfg file can be used to specify the files to be parsed for a list of IP addresses of devices that exist on the network.

Database tables used

This configuration file can be used to configure inserts into the following database tables:

- fileFinder.parseRules
- fileFinder.configuration

Note that there is another configuration file associated with the fileFinder database, the DiscoFileFinderSchema.cfg file, but you should not need to alter this file.

Sample: configuring the File finder to use five threads

The following example insert configures the File finder to use five threads.

```
insert into fileFinder.configuration
   ( m_NumThreads )
values
   ( 5 );
```

Example: configuring the File finder to parse /var/tmp/logged_hosts

The following example configuration instructs the File finder to parse an example text file, logged_hosts, that has been saved in the /var/tmp directory. The contents of the example file are shown below.

```
vi /var/tmp/logged_hosts

172.16.1.21   dharma              04:02:08
172.16.1.201  phoenix             19:07:08
172.16.1.25   lnd-sun-tivoli   15:10:00
172.16.2.33   ranger              19:07:07
~
"/var/tmp/logged_hosts" [Read only] 4 lines, 190 characters
```

The three columns in this example file respectively contain an IP address, the device name, and a time value. The columns are separated by white space, which can be multiple tabs, spaces, or a combination of both. You could configure the File finder to parse this example text file using an insert similar to the example.

```
insert into fileFinder.parseRules
(
            m_FileName, m_Delimiter, m_ColDefs
)
values
(
        "/var/tmp/logged_hosts",
        "[     ]+",
        [
            {
                m_VarName="m_UniqueAddress",
                m_ColNum=1
            },
            {
                m_VarName="m_Name",
                m_ColNum=2
            }
        ]
);
```

The above insert specifies that:

- The full path and name of the file is /var/tmp/logged_hosts.
- The source-file delimiter is white space. The column delimiter is indicated in the insert using a simple regular expression, [*tab space*]+ . You must press the **tab** and **space** keys rather than typing \t to represent the tab character.
- The first column contains IP addresses and must be mapped to the m_UniqueAddress column of the finders.returns table.
- The second column contains host names and must be mapped to the m_Name column of the finders.returns table.

Because the third column in the example text file is not relevant, it has not been mapped to a column of finders.returns and is ignored by the File finder during the discovery.

Example: configuring the File finder to parse the /etc/hosts file

The following insert instructs the File finder to:

- Parse /etc/hosts.
- Treat white space as the data separator.
- Use the following column definitions:
  - m_UniqueAddress for the first column
  - m_Name for the second column

```
insert into fileFinder.parseRules
(
            m_FileName,
            m_Delimiter,
            m_ColDefs
)
values
(
            "/etc/hosts",
            "[     ]",
            [
                {
                    m_VarName="m_UniqueAddress",
                    m_ColNum=1
                },
                {
                    m_VarName="m_Name",
                    m_ColNum=2
                }
            ]
);
```

Sample: configuring the File finder to parse /etc/defaultrouter

The following insert instructs the File finder to:

- Parse /etc/defaultrouter.
- Treat one or more occurrences of white space as the data separator.

- Use m_UniqueAddress as the column definition.

```
insert into fileFinder.parseRules
(
            m_FileName,
            m_Delimiter,
            m_ColDefs
)
values
(
            "/etc/defaultrouter",
            "[      ]+",
            [
                {
                    m_VarName="m_UniqueAddress",
                    m_ColNum=1
                }
            ]
);
```

Related reference:

"fileFinder database" on page 229
The fileFinder database defines the operation of the File finder.

DiscoHelperServerSchema.cfg configuration file

The `DiscoHelperServerSchema.cfg` configuration file defines the contents of the several helper databases.

Database tables used

This configuration file can be used to configure inserts into the following database tables.

ARP helper database tables:

- ARPHelper.ARPHelperTable
- ARPHelper.ARPHelperConfig

DNS helper database tables:

- DNSHelper.DNSHelperTable
- DNSHelper.DNSHelperConfig

Ping helper database tables:

- PingHelper.PingHelperTable
- PingHelper.PingHelperConfig

SNMP helper database tables:

- SnmpHelper.SnmpHelperTable
- SnmpHelper.SnmpHelperConfig

Telnet helper database tables:

- TelnetHelper.TelnetHelperTable
- TelnetHelper.TelnetHelperConfig

XMLRPC helper database tables:

- XmlRpcHelper.XmlRpcHelperTable
- XmlRpcHelper.XmlRpcHelperConfig

Related reference:

"The Helper Server databases" on page 233
When the Helper Server starts, it creates a database for each helper that is to be run.

DiscoPingFinderSeeds.cfg configuration file

The DiscoPingFinderSeeds.cfg configuration file is used for seeding the Ping finder and restricting device detection.

Database tables used

The DiscoPingFinderSeeds.cfg configuration file can be used to configure inserts into the following database tables:

- pingFinder.pingRules
- pingFinder.scope

Note that there is another configuration file associated with the pingFinder database, the DiscoPingFinderSchema.cfg file, but you should not need to alter this file.

Note: If you are seeding an IPv6 discovery, bear in mind that there are potentially billions of devices to be pinged within a single IPv6 subnet. To ensure that discovery completes, you must specify a sufficiently large netmask if you specify an IPv6 subnet as a ping seed.

Sample: seeding the Ping finder with a single device address

The following example insert defines a single seed with IP address of 10.10.2.224. This example does not specify values for m_NumRetries and m_TimeOut because they automatically take the default values from the configuration table.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not aupport an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

```
insert into pingFinder.pingRules
       ( m_Address, m_RequestType )
values
       ( "10.10.2.224", 1 );
```

Sample: seeding the Ping finder with a class B subnet address

The following example insert defines a single class B subnet as a seed.

```
insert into pingFinder.pingRules
       ( m_Address, m_RequestType, m_NetMask )
values
       ( "10.10.0.0", 2, "255.255.0.0" );
```

Sample: seeding the Ping finder with class C subnet addresses

The following example insert defines two Class 2 subnets as seeds.

```
insert into pingFinder.pingRules
       ( m_Address, m_RequestType, m_NetMask )
values
       ( "10.10.2.0", 2, "255.255.255.0" );
```

```
insert into pingFinder.pingRules
        ( m_Address, m_RequestType, m_NetMask )
values
        ( "10.10.47.0", 2, "255.255.255.0" );
```

Sample: restricting device detection

The following example insert configures the Ping finder to use the scope.zones table and use the discovery scope.

```
insert into pingFinder.scope
        ( m_UseScope, m_UsePingEntries )
values
        ( 1, 1 );
```

Important: Other combinations of m_UseScope and m_UsePingEntries filters are not recommended. Specifying values (0,0) results in an unbounded discovery, while specifying values (0,1) results in devices that you do not want to discover being needlessly pinged.

Related reference:

"pingFinder database" on page 230
The pingFinder database defines the operation of the Ping finder.

"IPv6 subnet mask sizes" on page 22
There are potentially billions of devices to be pinged within a single IPv6 subnet. To ensure that discovery completes, you must specify a sufficiently large netmask if you specify an IPv6 subnet as a ping seed.

DiscoPingHelperSchema.cfg configuration file

The DiscoPingHelperSchema.cfg configuration file defines how devices are to be pinged.

Database table used

The DiscoPingHelperSchema.cfg configuration file can be used to configure inserts into the pingHelper.configuration database table.

In this example configuration of the DiscoPingHelperSchema.cfg configuration file, the parameters specify to:

- Use 20 threads of process execution.
- Wait a maximum of 250 ms for a reply from a device.
- Retry unresponsive devices a maximum of five times.
- Wait 50 ms between pinging devices in a subnet.
- Not use broadcast or multicast pinging.

```
insert into pingHelper.configuration
(
        m_NumThreads,
        m_TimeOut,
        m_NumRetries,
        m_InterPingTime,
        m_Broadcast,
        m_Multicast
)
values
(
        20, 250, 5, 50, 0, 0
);
```

Related reference:

"Connectivity at the layer 3 network layer" on page 316
There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

"The Ping helper database" on page 249
The Ping helper database is defined by the DiscoPingHelperSchema.cfg configuration file. Its fully qualified database table name is pingHelper.configuration.

DiscoConfig.cfg configuration file

The DiscoConfig.cfg configuration file is used to have the Ping finder automatically check the devices discovered by the File finder, and to enable a context-sensitive discovery.

Database table used

The DiscoConfig.cfg configuration file can be used to configure inserts into the following tables:

- disco.config
- disco.managedProcesses
- disco.NATStatus
- disco.ipCustomTags
- disco.filterCustomTags
- translations.NATAddressSpaceIds
- translations.collectorInfo
- failover.restartPhaseAction
- failover.config
- failover.doNotCache

The following examples illustrate inserts into the disco.config database table.

Sample: pinging File finder devices

The following example command configures the discovery so that the devices discovered by the File finder are automatically checked by the Ping finder.

```
update disco.config set m_CheckFileFinderReturns = 1;
```

Sample: enabling a context-sensitive discovery

Attention: Enabling a context-sensitive discovery automatically enables all the Context agents. Disabling a context-sensitive discovery automatically disables all the Context agents. You should not manually enable or disable Context agents, either through the configuration files or through the Discovery Configuration GUI.

To enable a context-sensitive discovery, append the following insert to the `DiscoConfig.cfg` file:

```
insert into disco.config
(
     m_UseContext
)
```

```
values
(
        1
)
```

Inserting the value 0 disables the context-sensitive discovery.

Enriching topology using custom tags

You can use the disco.ipCustomTags and disco.filterCustomTags tables to enrich the discovered topology by associating one or more name-value pair tags with discovered entities.

Related concepts:

"Discovering device details (context-sensitive)" on page 288
The discovery of context-sensitive device details is carried out in several steps.

Related tasks:

"Adding tags to entities using custom tag tables" on page 173
You can add name-value pair tags to entities by creating inserts containing the name-value pair data into the disco.ipCustomTags table or into the disco.filterCustomTags table.

Related reference:

"Context-sensitive discovery agents" on page 329
There are several agents that take part in a context-sensitive discovery.

"disco.config table" on page 183
The config table configures the general operation of the discovery process.

DiscoScope.cfg configuration file

The DiscoScope.cfg configuration file can be used to configure the scope of a discovery.

Database tables used

This configuration file can be used to configure inserts into the following database tables:

- scope.zones
- scope.detectionFilter
- scope.instantiateFilter
- scope.special

Sample: defining an inclusion zone

The following example insert defines the 10.10.2.* subnet as an inclusion zone.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not support an IPv4–mapped IPv6 address such as ::ffff:192.0.2.128. Instead enter this address as ::ffff:c000:280 (standard colon-separated IPv6 format).

```
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
```

```
(
        1,
        1,
        [
            {
                m_Subnet="10.10.2.*"
            }
        ]
);
```

Sample: defining multiple inclusion zones

The following example defines three different IP inclusion zones each using a different syntax to define the subnet mask. The following devices are discovered:

- Any device within the 172.16.1.0 subnet (with a subnet mask of 24, that is, 24 bits turned on and 8 bits turned off, which implies a netmask of 255.255.255.0).
- Any device within the 172.16.2.0 subnet with a mask of 255.255.255.0.
- Any device within the 172.16.3.0 subnet with a mask of 255.255.255.0.

```
insert into scope.zones
(
    m_Protocol,
    m_Action,
    m_Zones
)
values
(
        1,
        1,
        [
              {
                    m_Subnet="172.16.1.0",
                    m_NetMask=24
              },
              {
                    m_Subnet="172.16.2.*"
              },
              {
                    m_Subnet="172.16.3.0",
                    m_NetMask=255.255.255.0
              }
        ]
);
```

Sample: defining an exclusion zone

The following example insert defines a single exclusion zone for the IP protocol, and associates the zone with a subnet.

```
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
(
        1,
        2,
        [
            {
                m_Subnet="172.16.1.0",
                m_NetMask=24
        ]
);
```

Sample: defining an inclusion zone within a NAT domain

The following example defines one inclusion zone. The inclusion zone includes any device with an IP address starting with 172.16.2 that also belongs to the NAT address space NATDomain1. The protocol is set to 1, that is, IP.

```
insert into scope.zones
(
    m_Protocol, m_Action, m_Zones, m_AddressSpace
)
values
(
        1,
        1,
        [
            {
                m_Subnet="172.16.2.*",
            }
        ],
        "NATDomain1"
);
```

Sample: restricting device interrogation based on IP address

The following example shows how to prevent the further interrogation of devices that match a given IP address. Only devices that do not have the IP address 10.10.63.234 are interrogated further.

There must be only one insert into the scope.detectionFilter table per protocol. Multiple conditions must be defined within a single insert.

Within the scope.detectionFilter table, specify:

- The type of network protocol. Currently only IP is supported.
- The filter condition(s). Only devices that pass this filter, that is, for which the filter evaluates true, are further investigated. If no filter is specified, all devices are passed through the detection filter.

```
insert into scope.detectionFilter
(
        m_Protocol, m_Filter
)
values
(
        1,
        "( ( m_UniqueAddress <> '10.10.63.234' ) )"
);
```

A stitcher tests each discovered device against the filter condition in the scope.detectionFilter table, and the outcome of this test determines whether the device is discovered.

Because the process flow of the discovery is fully configurable, you can configure this stitcher to act at any time during the discovery process. By default, the stitcher performs the conditional test on the device details returned by the Details agent. Your filter must therefore be based on the columns of the Details.returns table.

Although you can configure the filter condition to test any of the columns in the Details.returns table, you might need to use the IP address as the basis for the filter to restrict the detection of a particular device. If the device does not grant SNMP access to the Details agent, the Details agent might not retrieve MIB variables such as the Object ID. However, you are guaranteed the return of at least the IP address when the device is detected.

The following examples show how else you might configure the detection filter.

Sample: restricting interrogation based on Object ID

The following example shows how to prevent the further interrogation of devices that match a given Object ID. The OQL `not like` clause indicates that only devices that pass the filter (that is, devices for which the OID is *not* like 1.3.6.1.4.1.*) are interrogated further.

The backslash must be used in the insert to escape the `.`, which would otherwise be treated as a wildcard. A full explanation of the syntax of OQL can be found in the *IBM Tivoli Network Manager IP Edition Language Reference*.

```
insert into scope.detectionFilter
(
       m_Protocol,
       m_Filter
)
values
(
       1,
       "(
              ( m_ObjectId not like '1\.3\.6\.1\.4\.1\..*' )
       )"
);
```

Sample: combining multiple filter restrictions

You can combine filter conditions within a single OQL insert. The following example ensures that only devices that do not have the specified OID and do not have the specified IP address are detected:

```
insert into scope.detectionFilter
(
       m_Protocol,
       m_Filter
)
values
(
       1,
       "(
              ( m_ObjectId not like '1\.3\.6\.1\.4\.1\..*' )
              AND
              ( m_UniqueAddress <> '10.10.63.234' )
       )"
);
```

Restricting instantiation: limitation when filtering out interfaces

Note the following limitation when you restrict instantiation of interfaces.

Restriction: To ensure that alerts are not raised for *interfaces* that are excluded by the instantiation filter, you must set the `RaiseAlertsForUnknownInterfaces` variable. To this, perform the following steps:

1. Edit the `$NCHOME/etc/precision/NcPollerSchema.cfg` configuration file.
2. Add the following line to the file:

   ```
   update config.properties set RaiseAlertsForUnknownInterfaces = 1;
   ```

Sample: restricting instantiation based on the IP address

To restrict the devices that are instantiated, append an OQL insert into the scope.instantiateFilter table. There must be only one insert into the scope.instantiateFilter per protocol. The instantiateFilter table requires the following information:

- The type of network protocol. Currently only IP is supported.
- The conditional test. Only devices that pass the filter are sent to MODEL. If no filter is defined, all discovered devices are passed to MODEL.

The instantiateFilter works in the same way as the detectionFilter because a stitcher is called to compare discovered devices using the test defined in the scope.instantiateFilter table. By default, the test is performed after the Scratch Topology has been generated, but before the records are sent to MODEL. The conditional test must therefore be based on the columns of the scratchTopology.entityByName table.

Attention: You must ensure that there is only one insert into the scope.instantiateFilter table per protocol. Multiple filters must be combined within a single insert in the same way as is done in the detectionFilter table.

The following example shows how to restrict the instantiation of devices based on the IP address, by filtering against the m_Addresses column of the scratchTopology.entityByName table.

The m_Addresses column is a list of the OSI model layer 1-7 addresses for the device. The following example filter tests the value of m_Addresses(2), that is, the third entry in the list of addresses (list numbering starts at 0). The third entry in the list of addresses is the layer 3 address, that is, the IP address of the device.

The following insert ensures that only devices that pass the filter are instantiated, that is, devices for which the IP address is not 172.16.1.231, and is not 172.16.5.47, and does not begin 192.168.123.

You could also restrict instantiation based on other addresses for the device stored in the scratchTopology.entityByName.m_Addresses column. For example, m_Addresses(1) contains the layer 2 address of the device, that is, the MAC address.

```
insert into scope.instantiateFilter
(
        m_Protocol,
        m_Filter
)
values
(
        1,
        "(
                ( Address(2) <> "172.16.1.231" )
                AND
                ( Address(2) <> "172.16.5.47")
                AND
                ( Address(2) not like "192\.168\.123\..*" )
        )"
);
```

Sample: restricting instantiation based on Object ID

This example shows how to prevent the instantiation of devices that match a given Object ID. The OQL `not like` clause indicates that only devices that pass the filter (that is, devices for which the OID is not like 1.3.6.1.4.1.*) are instantiated.

```
insert into scope.instantiateFilter
(
        m_Protocol,
        m_Filter
)
values
(
        1,   // The backslash is used here to escape the .
        "(   // which would otherwise be treated
             // as a wildcard.
                ( EntityOID not like '1\.3\.6\.1\.4\.1\..*' )
        )"
);
```

Sample: complex instantiation restriction

You can configure a complex instantiation by combining conditions in the insert.

The following example shows a more complex insert, which combines a number of conditions relating to different columns of the scratchTopology.entityByName table.

```
insert into scope.instantiateFilter
(
        m_Protocol,
        m_Filter
)
values
(
        1,
        "(
                ( Address(2) = '10.82.219.1' )
                OR
                ( Address(2) = '10.82.213.5' )
                OR
                ( Address(2) = '10.82.213.6' )
        )
        OR
        (
                ( EntityName LIKE 'Tivoli' )
                AND
                ( EntityType < 3 )
        )
        OR
        (
                ( EntityType >= 3 )
                AND
                ( EntityType <> 7 )
        )"
);
```

The above insert ensures that only the following devices are sent to MODEL to be instantiated:

- Any device with the IP address 10.82.219.1, 10.82.213.5 or 10.82.213.6
- Any device that is not a Web server whose name contains the string `tivoli` (either lower-cased or capitalized) and for which `EntityType` < 3, that is, an interface or a chassis

- Any device of `EntityType` equal to 3, 4, 5, 6 or 8, that is, a logical interface, Virtual Local Area Network (VLAN) object, card, Power Supply Unit (PSU), or module

Related reference:

"Discovery scope database" on page 201
The scope database limits the extent or reach of a discovery. Using the scope database, you can configure a range of protocols and attributes that define zones that are to be included or excluded from the discovery process.

Devices with out-of-scope interfaces:

A network might contain devices that are within the discovery scope but that contain interfaces that are out of scope. Because the device is in scope, the default behavior of the layer 3 discovery agents is to download the interface table of the device and discover all the interfaces of a device, even if the interfaces themselves are out of scope.

If this situation applies to your network, and you want to modify the way in which the discovery process handles devices that are partially in scope, there are several ways to modify the discovery and monitoring process to exclude these interfaces from the discovery.

A possible configuration adjustment is to modify the insert into the scope.instantiateFilter such that the out-of-scope interfaces are not instantiated. This solution means that the out-of-scope interfaces are still discovered, but are not passed to MODEL to be instantiated to an Active Object Class (AOC); therefore the out-of-scope interfaces are not represented on the topology or monitored.

DiscoSnmpHelperSchema.cfg configuration file

The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

Database table used

The DiscoSnmpHelperSchema.cfg configuration file can be used to configure inserts into the snmpHelper.configuration database table.

You can also configure the SNMP Helper to use the GetBulk operation when SNMP v2 or v3 is used. Use of the GetBulk operation improves discovery speed. For more information, see the *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*.

Sample: Configuring timeouts and threads

The following example configuration causes the SNMP helper to behave as follows:

- 120 threads of program execution are started to process incoming requests for SNMP data from the Helper Server. The SNMP helper processes a maximum of 120 such requests simultaneously.
- A three-second timeout period is specified for a device to respond to an SNMP query issued by the SNMP helper. If the device has not responded after this time, the helper issues the request again, one time.

```
insert into snmpHelper.configuration
(
        m_NumThreads,
        m_TimeOut,
        m_NumRetries,
)
values
(
    120, 3000, 1
);
```

Related reference:

"Discovering connectivity among Ethernet switches" on page 311
Discovery agents that discover connectivity information between Ethernet switches have three main operational stages: gain access to the switch and download all the switch interfaces; discover VLAN information for the switch; download the forward database table for the switch.

"Discovering connectivity among ATM devices" on page 321
Asynchronous Transfer Mode (ATM) is an alternative switching protocol for mixed format data (such as pure data, voice, and video). Several types of discovery agents can be used to discover ATM devices on a network.

"Discovering NAT gateways" on page 324
There are several agents that download Network Address Translation (NAT) information from known NAT gateways.

"Discovering containment information" on page 325
An important principle used by the network model is containment. A container holds other objects. You can put any object within a container and even mix different objects within the same container.

"Discovery agents on other protocols" on page 327
Network Manager provides agents that discover devices that use other protocols than ones previously described.

"Task-specific discovery agents" on page 330
There is a group of discovery agents that are task-specific.

"The SNMP helper database" on page 250
The SNMP helper database is defined by the DiscoSnmpHelperSchema.cfg configuration file. Its fully qualified database table name is snmpHelper.configuration.

DiscoTelnetHelperSchema.cfg configuration file

The DiscoTelnetHelperSchema.cfg configuration file defines the operation of the Telnet helper, which returns the results of a Telnet operation into a specified device.

Database tables used

The DiscoTelnetHelperSchema.cfg configuration file can be used to configure inserts into the following database tables:

- telnetHelper.configuration
- telnetHelper.deviceConfig

You can configure the Telnet helper to use the Secure Shell (SSH) program. SSH enables authentication and provides more secure communications over the network.

Sample: configuring the Telnet helper

The following insert can be appended to the DiscoTelnetHelperSchema.cfg configuration file to configure the operation of the Telnet helper. The insert configures the Telnet helper to:

- Use 20 threads of process execution

- Wait a maximum of 5000 ms for a reply from a device
- Try the request up to three times

```
insert into telnetHelper.configuration
(
        m_NumThreads,
        m_TimeOut,
        m_Retries
)
values
(
        20,
        5000,
        3
);
```

Configuring device-specific settings

The Telnet helper also accepts multiple inserts into the telnetHelper.deviceConfig table within the DiscoTelnetHelperSchema.cfg configuration file that define the interaction of the Telnet operation.

The following examples show how to configure Telnet device-specific settings. You can configure device settings based on the sysObjectID MIB variable or based on IP or subnet. The most effective way to set these options is based on the sysObjectID MIB variable. This variable identifies the device vendor. Device-specific configuration options typically vary with the device vendor. You can configure values for all Cisco devices, for example, regardless of where these devices are in the network.

Sample: configuring settings for devices from a specific vendor

The following typical configuration shows how to configure settings for all devices from a specific vendor. The insert specifies:

- 1.3.6.1.4.1.9.1. as the sysObjectID MIB variable to match for this configuration entry. All devices with object IDs of the form 1.3.6.1.4.1.9.1.* are matched. In general, these are Cisco IOS devices, although there are exceptions.
- terminal length is the command that sets the output page length for Cisco devices.

  Note: This command varies with devices of different vendor types.
- No paging
- Prompt from remote device
- The response to send to the remote device for it to continue paged output.

```
insert into telnetHelper.deviceConfig
(
        m_SysObjectId,
        m_PageLengthCmd,
        m_PageLength,
        m_ContinueMsg,
        m_ContinueCmd
)
values
(
        "1.3.6.1.4.1.9.1.", "terminal length", 0, ".*[Mm]ore.*", " "
);
```

The DiscoTelnetHelperSchema.cfg configuration file contains inserts with default device-specific configuration settings for the following vendor types:

- Cisco IOS devices
- Cisco Cat OS devices
- Juniper JUNOS devices
- Juniper ERX devices
- Huawei devices
- Dasan devices

Sample: configuring device response settings based on IP address

If the output of the telnet command is longer than one page, the device sends a message asking whether to display the next page. Configure the messages to be expected and the responses to be given by the Telnet helper in the DiscoTelnetHelperSchema.cfg configuration file.

Commands beginning m_Continue (such as m_ContinueMsg) and m_PageLength (such as m_PageLengthCmd) are mutually exclusive: you must use one or the other. If these settings are not configured correctly for your devices, data might be lost.

The following example shows how to configure settings for devices based on the IP address. The insert specifies:

- 192.168.112.0 as the IP address
- The prompt from the remote device is a regular expression containing "`wish to continue`"
- The response to send to the remote device for it to continue paged output is "y"

```
insert into telnetHelper.deviceConfig
(
         m_IpOrSubNet,
         m_NetMaskBits,
         m_Protocol,
         m_ContinueMsg,
         m_ContinueCmd
)
values
(
         192.168.112.0,
         24,
         1,
         ".*wish to continue.*",
         "y"
);
```

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

"Discovering connectivity among Ethernet switches" on page 311
Discovery agents that discover connectivity information between Ethernet switches have three main operational stages: gain access to the switch and download all the switch interfaces; discover VLAN information for the switch; download the forward database table for the switch.

"Discovering NAT gateways" on page 324
There are several agents that download Network Address Translation (NAT) information from known NAT gateways.

"Context-sensitive discovery agents" on page 329
There are several agents that take part in a context-sensitive discovery.

"The Telnet helper database" on page 251
The Telnet helper database is defined by the DiscoTelnetHelperSchema.cfg configuration file. Its fully qualified database table names are: telnetHelper.configuration; telnetHelper.deviceConfig.

DiscoXmlRpcHelperSchema.cfg configuration file

The DiscoXmlRpcHelperSchema.cfg configuration file can be used to configure the XML-RPC helper, which enables Network Manager to communicate with EMS collectors using the XML-RPC interface.

Database table used

The DiscoXmlRpcHelperSchema.cfg configuration file can be used to configure inserts into the xmlRpcHelper.configuration database table.

This example insert configures the XML-RPC helper to:

- Use one thread of process execution.
- Allow a maximum size of 1048576 bytes for an XML-RPC response.

```
insert into xmlRpcHelper.configuration
(
    m_NumThreads,
    m_MaxResponseSize
)
values
(
    1, 1048576
);
```

Note: The default maximum response size might be too small when running a Collector-based discovery against Collectors that result in very large responses. In such cases, increase the maximum response size. To increase the maximum response size, set the **m_MaxResponseSize** parameter to a higher value. Make sure you set the same value for **m_MaxResponseSize** in both of the following files:

- NCHOME/etc/precision/DiscoCollectorFinderSchema.cfg
- NCHOME/etc/precision/DiscoXmlRpcHelperSchema.cfg

Related reference:

"The XMLRPC helper database" on page 252
The XMLRPC helper database is defined by the DiscoXmlRpcHelperSchema.cfg configuration file. Its fully qualified database table name is xmlRpcHelper.configuration.

SnmpStackSecurityInfo.cfg configuration file

The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

Database tables used

This configuration file can be used to configure inserts into the following database tables:

- snmpStack.configuration
- snmpStack.verSecurityTable
- snmpStack.accessParameters

Note that there is another configuration file associated with the snmpStack database, the SnmpStackSchema.cfg file, but you should not need to alter this file.

You can also configure the SNMP Helper to use the GetBulk operation when SNMP v2 or v3 is used. Use of the GetBulk operation improves discovery speed. For more information, see the *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*.

Sample: Configuring SNMP versions

If auto-versioning is turned on, the following configuration adjustment specifies that a community string of 'public' is used for devices that support SNMP version 1, and specific configuration is used for devices that support SNMP version 3. Since no value has been specified for m_SnmpPort, this value defaults to the standard SNMP 161 port.

```
insert into snmpStack.verSecurityTable
(
        m_SNMPVersion,
        m_Password,
        m_SNMPVer3Level,
        m_SNMPVer3Details,
        m_SecurityName,
)
values
(
        0,
        'public',
        2,
        {
            m_AuthPswd="authpassword",
            m_PrivPswd="privpassword"
        },
        'authPriv'
);
```

Sample: Defining community strings

The following inserts define the community strings `public` and `crims0n` for use to access SNMP devices.

You can append as many inserts as there are passwords to the `SnmpStackSecurityInfo.cfg` configuration file. All password and subnet configurations are tried until a match is found.

Note: Only one SNMP community string, the `public` community string, is set up by default.

```
insert into snmpStack.verSecurityTable
(
        m_SNMPVersion,
        m_Password,
        m_SNMPVer3Level,
        m_SNMPVer3Details,
        m_SecurityName
)
values
(
        0,
        'public',
        2,
        {
            m_AuthPswd="authpassword",
            m_PrivPswd="privpassword"
        },
        'authPriv'
);

insert into snmpStack.verSecurityTable
(        m_IpOrSubNetVer,
        m_NetMaskBitsVer,
        m_SNMPVersion,
        m_Password,
        m_SNMPVer3Level,
        m_SNMPVer3Details,
        m_SecurityName
)
values
(
        "10.10.2.0",
        24,
        0,
        'crims0n',
        2,
        {
            m_AuthPswd="authpassword",
            m_PrivPswd="privpassword"
        },
        'authPriv'
);
```

Sample: Specifying an SNMP port

This example configures the same SNMP settings as in the previous example on all devices within the subnet 192.168.64.0 and specifies the SNMP port as 6161 on all devices within this subnet.

```
insert into snmpStack.verSecurityTable
(
        m_IpOrSubNetVer,
        m_NetMaskBitsVer,
        m_SNMPVersion,
        m_Password,
        m_SNMPVer3Level,
        m_SNMPVer3Details,
        m_SecurityName,
        m_SnmpPort,
)
values
(
        192.168.64.0,
        24,
        0,
        'public',
        2,
        {
            m_AuthPswd="authpassword",
            m_PrivPswd="privpassword"
        },
        'authPriv'
        6161
);
```

Related reference:

"Discovering connectivity among Ethernet switches" on page 311
Discovery agents that discover connectivity information between Ethernet switches have three main operational stages: gain access to the switch and download all the switch interfaces; discover VLAN information for the switch; download the forward database table for the switch.

"Types of agents" on page 311
The agents supplied with Network Manager can be divided into categories according to the type of data they retrieve or the technology they discover.

"Connectivity at the layer 3 network layer" on page 316
There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

"Discovering connectivity among ATM devices" on page 321
Asynchronous Transfer Mode (ATM) is an alternative switching protocol for mixed format data (such as pure data, voice, and video). Several types of discovery agents can be used to discover ATM devices on a network.

"Discovering NAT gateways" on page 324
There are several agents that download Network Address Translation (NAT) information from known NAT gateways.

"Discovering containment information" on page 325
An important principle used by the network model is containment. A container holds other objects. You can put any object within a container and even mix different objects within the same container.

"Discovery agents on other protocols" on page 327
Network Manager provides agents that discover devices that use other protocols than ones previously described.

"Task-specific discovery agents" on page 330
There is a group of discovery agents that are task-specific.

"snmpStack database" on page 211
The snmpStack database defines the operation of the SNMP helper.

TelnetStackPasswords.cfg configuration file

The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

You can use the TelnetStackPasswords.cfg configuration file to specify a Secure Shell (SSH) connection when configuring Telnet device access. SSH enables password encryption when performing Telnet access. SSH versions 1 and 2 are supported (restrictions apply in FIPS mode).

Important: SSH within Network Manager IP Edition currently supports password-based authentication or no authentication. It does not support RSA signature authentication.

Database table used

The TelnetStackPasswords.cfg configuration file can be used to configure inserts into the telnetStack.passwords database table.

Note that there is another configuration file associated with the telnetStack database, the TelnetStackSchema.cfg file, but you should not need to alter this file.

Sample: Configuring Telnet access parameters for a subnet

The following example insert configures the Telnet access parameters for a subnet. The insert specifies:

- A subnet address of 192.168.200.0 with a netmask of 25.
- The password and username to use to access the device.
- The password, login and console prompts to expect from the device.
- The devices on this subnet support SSH.

```
insert into telnetStack.passwords
(
        m_IpOrSubNet,
        m_NetMaskBits,
        m_Password,
        m_Username,
        m_PwdPrompt,
        m_LogPrompt,
        m_ConPrompt,
        m_SSHSupport
)
values
(
        '192.168.200.0',
        25,
        '3v3rt0n',
        'user',
        '.*assword:.*',
        '.*ogin.*',
        '.*onsole>.*',
        1
);
```

Sample: Configuring Telnet access parameters for a device

The following example insert shows how you can configure the access parameters for a single IP address. The insert specifies:

- A single IP address of 172.16.1.21. The address is identified as a single address by the fact that m_NetMaskBits=32.

- The password and username to use to access the device.
- The password, login and console prompts to expect from the device.
- This device does not support SSH.

```
insert into telnetStack.passwords
(
        m_IpOrSubNet,
        m_NetMaskBits,
        m_Password,
        m_Username,
        m_PwdPrompt,
        m_LogPrompt,
        m_ConPrompt,
        m_SSHSupport
)
values
(
        '172.16.1.21',
        32,
        '',
        '',
        '.*assword.*',
        '.*sername.*',
        '.*Morr.*',
        0
);
```

Sample: Configuring Telnet device-access for a subnet

The following example insert configures the Telnet access parameters for a subnet. The insert specifies:

- A subnet address of 192.168.200.0 with a netmask of 25.
- The password and username to use to access the device.
- The password, login and console prompts to expect from the device.
- The devices on this subnet support SSH.

```
insert into telnetStack.passwords
(
        m_IpOrSubNet,
        m_NetMaskBits,
        m_Password,
        m_Username,
        m_PwdPrompt,
        m_LogPrompt,
        m_ConPrompt,
        m_SSHSupport
)
values
(
        '192.168.200.0',
        25,
        '3v3rt0n',
        'user',
        '.*assword:.*',
        '.*ogin.*',
        '.*onsole>.*',
        1
);
```

Sample: Configuring Telnet device-access for a single IP address

The following example insert shows how you can configure the access parameters for a single IP address. The insert specifies:

- A single IP address of 172.16.1.21. The address is identified as a single address by the fact that m_NetMaskBits=32.
- The password and username to use to access the device.
- The password, login and console prompts to expect from the device.
- This device does not support SSH.

```
insert into telnetStack.passwords
(
        m_IpOrSubNet,
        m_NetMaskBits,
        m_Password,
        m_Username,
        m_PwdPrompt,
        m_LogPrompt,
        m_ConPrompt,
        m_SSHSupport
)
values
(
        '172.16.1.21',
        32,
        '',
        '',
        '.*assword.*',
        '.*sername.*',
        '.*Morr.*',
        0
);
```

Related reference:

"Discovering connectivity among Ethernet switches" on page 311
Discovery agents that discover connectivity information between Ethernet switches have three main operational stages: gain access to the switch and download all the switch interfaces; discover VLAN information for the switch; download the forward database table for the switch.

"Types of agents" on page 311
The agents supplied with Network Manager can be divided into categories according to the type of data they retrieve or the technology they discover.

"Connectivity at the layer 3 network layer" on page 316
There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

"Discovering NAT gateways" on page 324
There are several agents that download Network Address Translation (NAT) information from known NAT gateways.

"Context-sensitive discovery agents" on page 329
There are several agents that take part in a context-sensitive discovery.

"telnetStack database" on page 215
The telnetStack database defines the Telnet access parameters for devices.

Retrieving extra information

You can configure the discovery agents to retrieve extra information from devices and store this information in the ExtraInfo column of the topology database.

To specify that extra information be retrieved by a given discovery agent, modify the definition file of the agent ($NCHOME/precision/disco/agents/*.agnt). All discovery agents have a definition file in the agents directory, regardless of whether the agent is text-based or precompiled.

The changes that you must make to the agent definition are described in the following topics.

Changing the agent type

You can change the agent type in the agent definition file.

At the start of the discovery agent definition file, one of the following types of agent is identified:

- `DiscoCompiledAgent{}`: Denotes a compiled discovery agent (with a corresponding shared library in the `$NCHOME/precision/lib` directory).
- `DiscoDefinedAgent{}`: Denotes a text-based discovery agent (with no corresponding shared library).
- `DiscoCombinedAgent{}`: Denotes a discovery agent that is a combination of text-based and precompiled, where extra processing (such as the retrieval of extra information from devices) is defined in the discovery agent definition file.

To retrieve extra information from devices, the agent type must either be `DiscoDefinedAgent{}` or `DiscoCombinedAgent{}`. Therefore, if you are modifying an existing compiled agent to retrieve extra information, the first step is to change the type of agent from `DiscoCompiledAgent{}` to `DiscoCombinedAgent{}`.

Mediation and processing layers

The retrieval of extra information from devices and the addition of the information to the entity records is conducted in two layers: the mediation and processing layers. In the mediation layer, the actual SNMP requests to retrieve the variables are carried out. In the processing layer, the retrieved variables are added to the appropriate entity records. There is also an optional filter on the mediation layer.

The following code segment is an overview of the structure of the mediation and processing sections of the discovery agent definition file.

```
DiscoAgentMediationFilter
            {
                    // Optional section containing filters for the mediation layer.
            }

DiscoAgentMediationLayer
            {
                    // Contains the SNMP Get and GetNext requests to be performed.
                    // In addition, an ICMP trace can be performed and SNMP access
                    // parameters can be retrieved in the mediation layer.
            }

DiscoAgentProcessingLayer
            {
                    // Adds the retrieved variables to the appropriate entity
                    // record(s).
            }
```

The mediation layer

The mediation layer is where the SNMP and ICMP requests are performed.

In the following code, the `DiscoSnmpGetResponse();` rule performs an SNMP Get request, and the `DiscoSnmpGetNextResponse();` rule performs an SNMP Get Next request. You can include as many of each type of request as necessary.

You can also include the `DiscoSnmpGetAccessParameters();` rule, which retrieves the SNMP access details for the device, and the `DiscoICMPGetTrace();` rule, which retrieves all the IP addresses in the path to the device.

```
DiscoAgentMediationLayer
  {
         DiscoSnmpRequests
                {
                        DiscoSnmpGetResponse( ARGUMENT, VARIABLE );
                        DiscoSnmpGetNextResponse( ARGUMENT, VARIABLE, );
                        DiscoSnmpGetAccessParameters( VARIABLE );
                }
         DiscoICMPRequests
                {
                        DiscoICMPGetTrace( VARIABLE );
                }
  }
```

DiscoSnmpGetResponse();:

DiscoSnmpGetResponse(); performs an SNMP Get request. The simple form of this rule takes two arguments, separated by a comma. The first argument is the key to assign to the response. This key is used in the processing layer. The second argument is the OID (Object ID) to retrieve from the device.

The following example retrieves sysUpTime, assigning the key m_SysUpTime to the value that is returned.

```
DiscoSnmpGetResponse( "m_SysUpTime", sysUpTime );
```

A more complex form of DiscoSnmpGetResponse(); takes a third argument, the OID index. The following example retrieves ifDescr, assigns the key m_IfDescr to the value returned, and uses the OID index 1.

```
DiscoSnmpGetResponse( "m_IfDescr", ifDescr, "1" );
```

DiscoSnmpGetNextResponse();:

DiscoSnmpGetNextResponse(); performs an SNMP GetNext request. This rule takes the same arguments as DiscoSnmpGetResponse();.

The following example retrieves ipRouteIfIndex and assigns the key m_IpRouteIfIndex to the value returned.

```
DiscoSnmpGetNextResponse( "m_IpRouteIfIndex", ipRouteIfIndex );
```

DiscoSnmpGetAccessParameters();:

`DiscoSnmpGetAccessParameters();` retrieves the SNMP access parameters for the device.

If you configure the discovery agent to retrieve the access parameters in the mediation layer, you must also configure the agent to add the information to the database record in the processing layer.

```
DiscoSnmpGetAccessParameters( "m_AccessParam" );
```

DiscoICMPGetTrace();:

`DiscoICMPGetTrace();` retrieves the IP addresses in the path to the device.

If you configure the discovery agent to retrieve the path to the device in the mediation layer, you must also configure the agent to add the information to the database record in the processing layer.

```
DiscoICMPGetTrace( "m_Trace" );
```

Mediation layer filter

The mediation layer filter is an optional filter that restricts the SNMP requests for extra information to specific devices. You can include a condition within the DiscoMediationSnmpGetFilter{} section within the DiscoAgentMediationFilter{}, that only devices passing the filter are processed by the agent.

The following example ensures that only devices with an ipForwarding value of 1 are processed.

```
DiscoAgentMediationFilter
            {
                    DiscoMediationSnmpGetFilter
                    {
                            "ipForwarding" = 1 ;
                    }
            }
```

The processing layer

The processing layer is where the retrieved information is added to the entity records. Both the DiscoAgentProcLayerAddTags{} and DiscoAgentProcLayerAddLocalTags{} sections are optional. However, if both sections are omitted, no extra information is stored in the database records.

The structure of the processing layer is shown below.

```
DiscoAgentProcessingLayer
  {
       DiscoAgentProcLayerAddTags
               {
                    DiscoAddTagSnmpGet( KEY );
                    DiscoAddTagSnmpGetNext( KEY );
                    DiscoAddTagSnmpGetAccessParameters( "m_AccessParam" );
                    DiscoAddTagTrace( "m_Trace" );
               }
       DiscoAgentProcLayerAddLocalTags
               {
                    DiscoAddTagSnmpGet(
                                    TAG FROM KEY WHERE CONDITION );
                    DiscoAddTagSnmpGetNext(
                                    TAG FROM KEY WHERE CONDITION );
               }
  }
```

DiscoAgentProcLayerAddTags{}:

Within the DiscoAgentProcLayerAddTags{} section, you can include as many DiscoAddTagSnmpGet(); or DiscoAddTagSnmpGetNext(); rules as necessary. These rules add the retrieved variable to the database record for the discovered entity.

Each rule within the DiscoAgentProcLayerAddTags{} section takes a single argument, which is the key assigned to the retrieved variable in the mediation layer. The following example adds the value of m_SysUpTime, retrieved in the mediation layer, to the entity record.

```
DiscoAddTagSnmpGet( "m_SysUpTime" );
```

If you configured the discovery agent to retrieve either the SNMP access parameters or the path to the device during the mediation layer, you must include either the DiscoAddTagSnmpGetAccessParameters(); or the DiscoAddTagTrace(); rule in the DiscoAgentProcLayerAddTags{} section to ensure that the retrieved information is added to the MODEL database.

DiscoAgentProcLayerAddLocalTags{}:

Within the DiscoAgentProcLayerAddLocalTags{} section, you can include as many DiscoAddTagSnmpGet(); or DiscoAddTagSnmpGetNext(); rules as necessary. These rules add the retrieved variable to the database record for a local neighbor.

The structure of the rules is:

```
DiscoAddTagSnmpGet( TAG FROM KEY WHERE CONDITION );
DiscoAddTagSnmpGetNext( TAG FROM KEY WHERE CONDITION );
```

The arguments that determine the local neighbor to which the tag is added are:

- *TAG* specifies the field name of the tag to be added.
- *KEY* indicates the key assigned to the value returned in the mediation layer.
- *CONDITION* indicates a condition that determines whether or not the tag is added.

The following example adds a field called m_IfDescr to the local neighbor object (using the value returned in the mediation layer that was assigned the key m_IfDescr) where m_IfIndex=1.

```
DiscoAddTagSnmpGet( "m_IfDescr" FROM "m_IfDescr"
                                WHERE ( "m_IfIndex" = "1" )
                             );
```

The following example adds a field called m_IfType to the local neighbor object using the list of values returned by the GetNext request performed in the mediation layer and assigned the key m_IfType. The WHERE clause indicates the particular value required from the list of data. The value is retrieved by looking for the entry where the value of the m_IfIndex field in the local neighbor object is equal to SNMPINDEX(0), that is, the first value of the SNMP table entry.

```
DiscoAddTagSnmpGetNext( "m_IfType" FROM "m_IfType"
                                WHERE ( "m_IfIndex" = SNMPINDEX(0) )
                            );
```

Special case: adding information to the master.entityByNeighbor table

You can configure a discovery agent to download MIB variables, and specify that the variables populate the MODEL `master.entityByNeighbor` table.

If you configure a discovery agent to download any of the MIB variables listed in Table 4 and add those variables to the entity under the corresponding names, then they are used to populate the corresponding columns in the MODEL `master.entityByNeighbor` table. These columns can only be populated for entities that contain the `RelatedTo` field, that is, entities that are related to other entities.

Table 4. Variables used to populate the master.entityByNeighbor table

MIB variable	Name under which variable must be configured to be added to entity	Column to be populated
ifSpeed	m_IfSpeed	Speed
ifRelType	m_IfRelType	RelType
ifProtocol	m_IfProtocol	Protocol

Administering traps

The SNMP trap multiplexer, the **`ncp_trapmux`** process, listens to a single port and forwards all the traps it receives to a set of host/socket pairs.

The following topics describe how to administer traps.

About trap management

Trap management enables you to ensure that traps received from network devices are forwarded to ports where they can be handled by Network Manager and other network management systems.

In most networks, traps arrive on a single default port (usually port 162). This can cause problems if you have Network Manager and another network management system installed on the same server. Both of these systems might need to listen for traps; however, only one process can bind to one port at a time.

The SNMP Trap Multiplexer is a Network Manager process that resolves this problem: it listens to a single port and forwards all the traps it receives to a set of host/socket pairs.

By default, the SNMP Trap Multiplexer listens for traps on port 162, but you can change this by inserting an alternative port number into the trapMux.config database table.

The ncp_trapmux process can also store trap events in a binary format file (containing trap and timing information) that can be used to recreate the trap events in the order they occurred at a later date. This is useful mainly for debugging purposes.

Starting the SNMP trap multiplexer

Although it is good practice to ensure that the **ncp_ctrl** process is configured to launch and manage the SNMP Trap Multiplexer, you can also start it manually.

To start the **ncp_trapmux** process, use the following command:

```
ncp_trapmux -domain DOMAIN_NAME
```

Forwarding traps

Using the SNMP Trap Multiplexer, you can forward traps to one or more servers.

To configure the SNMP Trap Multiplexer to forward traps to network management systems running on host1 and host2:

1. Edit the schema file, `$NCHOME/etc/precision/TrapMuxSchema.cfg`, to contain a set of host/socket pairs. For example, append the following lines to the file:

   ```
   insert into trapmux.sinkHosts (host, port) values ("host1", 5999);
   insert into trapmux.sinkHosts (host, port) values ("host2", 5999);
   ```

2. Start the SNMP Trap Multiplexer using the following commands:

   ```
   ncp_trapmux -domain DOMAIN1
   ncp_trapmux -domain DOMAIN2
   ```

In the above example, when a trap is sent to the server that is running the **ncp_trapmux** process, it is forwarded to `test-host1`, port `5999` and `test-host2`, port `5999`.

Starting trap capture:

You can start capturing traps by inserting commands into the SNMP Trap Multiplexer database.

To instruct the SNMP Trap Multiplexer to start capturing traps:

1. Log into the `TrapMux` service using the OQL Service Provider or the Management Database Access page.
2. Issue the following commands:

   ```
   insert into trapMux.command
   (command) values( "capture_start" );
   go
   ```

Stopping trap capture:

You can stop capturing traps by inserting commands into the SNMP Trap Multiplexer database.

To instruct the SNMP Trap Multiplexer to stop capturing traps:

1. Log into the `TrapMux` service using the OQL Service Provider or the Management Database Access page.
2. Issue the following commands:

   ```
   insert into trapMux.command
   (command) values( "capture_stop" );
   go
   ```

Printing traps to a file:

You can print traps to a file by inserting commands into the SNMP Trap Multiplexer database.

To instruct **ncp_trapmux** to print traps:

1. Log into the TrapMux service using the OQL Service Provider or the Management Database Access page.
2. Issue the following commands:

   ```
   insert into trapMux.command
   (command, fileName) values( "print", FILENAME );
   go
   ```

Where *FILENAME* specifies the file to which the output is written. If the file is not specified, $NCHOME/etc/precision/trapmux.out is used.

Replaying traps from a file:

If you have created a text-readable file for traps, you can use the **ncp_trapmux** process to recreate the trap events in the order specified in this file.

The **ncp_trapmux** process can replay traps using a binary file or a human-readable file, however, the **ncp_trapmux** process can only generate binary files.

To instruct **ncp_trapmux** to replay traps from a file:

1. Log into the TrapMux service using the OQL Service Provider or the Management Database Access page.
2. Issue the following commands:

   ```
   insert into trapMux.command
   (command, fileName) values( "replay", "trapmux.out" );
   go
   ```

SNMP trap multiplexer commands

You can issue commands to the SNMP trap multiplexer, the ncp_trapmux process, to control its operation.

The commands used to control the ncp_trapmux process are described in the following table:

Table 5. Commands used to control the ncp_trapmux process

Command	**Function and Default Filename**
capture_start	Begin logging traps to memory. The default filename is NULL (not required).
capture_stop	Stop logging traps to memory. The default filename is NULL (not required).
capture_continue	Continue logging traps to memory. The default filename is NULL (not required).
capture_empty	Clear memory of all currently logged traps. The default filename is NULL (not required).
rehash	Shut down the ncp_trapmux process and clear all memory. The daemon then rereads the configuration file and starts up again. The default filename is NULL (not required).
restart	Set the daemon to normal mode. The default filename is NULL (not required).

Table 5. Commands used to control the ncp_trapmux process (continued)

Command	Function and Default Filename
`replay`	Either read the traps in memory or read the raw trap packet information in the specified file and replay the traps with a small delay between each. The default filename is NULL (play from memory).
`replay timed`	Either read the traps in memory or read the raw trap packet information in the specified file and replay the traps in the order of the time they were received with the same delays between traps. The default filename is NULL (play from memory).
`print`	Print the current traps in memory in a non-readable format to the specified file. Time information is encoded with the trap. The default filename is `$NCHOME/etc/precision/trapmux.out`.

Configuring specialized discoveries

You can configure the system to perform more complex discoveries, such as MPLS and NAT discovery.

Specialized discoveries include the following:

EMS discoveries
Collects topology data from Element Management Systems and integrates this data into the discovered topology.

MPLS discoveries
Discovers layer 3 VPNs and enhanced layer 2 VPNs running across MPLS core networks.

NAT discoveries
Discovers NAT gateway devices thereby enables you to retrieve data on devices in private address spaces.

Configuring EMS discoveries

You can configure Network Manager to collect topology data from Element Management Systems (EMS) and integrate this data into the discovered topology.

The following topics describe how to configure an EMS discovery.

For an overview of how Network Manager collects topology data from Element Management Systems (EMSs) and integrates this data into the discovered topology, see the *IBM Tivoli Network Manager IP Edition Product Overview*.

Related concepts:

"Discovery process with EMS integration" on page 295
Network Manager collects topology data from an EMS using collectors.

About EMS integration

The Network Manager EMS integration enables Network Manager to collect topology data from Element Management Systems.

Table 6 shows the steps involved in collecting topology data from EMS as part of a discovery or partial discovery. After this data has been collected, Network Manager stitches it together with the topology.

Table 6. Collecting topology data from EMS during discovery

Step	Data Flow
1	Using the Collector finder, the discovery system queries the collector to obtain a list of devices managed by the EMS. In the case of a partial discovery, the discovery might query for a single device or subnet only.
2	The Collector queries the EMS for the list of devices.
3	The EMS responds with list of managed devices.
4	The Collector responds by providing the list of devices.
5	Using a number of specialized collector discovery agents at different times during the discovery, the discovery system queries the collector for basic and detailed information about each of the devices in the list. Detailed information requested includes inventory information, layer 2 and layer 3 connection details, and VPN information.
6	The Collector responds by providing basic and detailed information as this is requested.

About collectors:

A collector is a software module that retrieves topology data from a data source, such as an Element Management System (EMS) or a comma-separated value (CSV) file, and makes this data available to the discovery process as a set of XML data. Network Manager can then stitch this data into the discovered topology.

A collector translates the topology data from the format in which it is held in the proprietary EMS into a standard XML structure that can be processed by Network Manager. This means that a different collector must be developed for each different EMS vendor and model. Network Manager ships with a collector that processes data from an Alcatel 5620 SAM EMS. This collector is written in the Perl language. Collectors may be written in any language. However, Network Manager ships with Perl modules to support the development of Perl-based collectors.

Collectors can run on the same host as the Network Manager. Collectors can also run on a separate host.

All interaction between Network Manager and the collectors is conducted using XML, and this interaction occurs over an XML-RPC interface.

Related information:

Tivoli Field Guide: EMS Collector Developer Guide

Network Manager ships with an Alcatel 5620 SAM EMS collector that is ready for immediate use.See the EMS Collector Developer Guide for information on how to develop collectors for other Element Management Systems.

Default collectors:

A number of collectors are supplied with Network Manager.

Alcatel5620SamSoap

Collector for the Alcatel 5620 SAM EMS. This collector uses the SOAP access protocol to communicate with the EMS.

Alcatel5620SamSoapFindToFile

Collector for the Alcatel 5620 SAM EMS. This collector uses the SOAP access protocol to communicate with the EMS. The collector retrieves the same data as the Alcatel5620SamSoapFindToFile collector.

The collector stores the data from the EMS in XML files with the same name as the objects queried. The collector transfers the XML files to the Network Manager using FTP. You must configure the FTP connection details before running the collector.

Alcatel5620SamCsv

Collector for the Alcatel 5620 SAM EMS. This collector retrieves EMS topology data from a CSV dump of the Alcatel 5620 SAM EMS.

Alcatel5529IdmSoap

Collector for the Alcatel-Lucent 5529 Inventory Data Manager (IDM) EMS. The collector retrieves containment information for devices managed by the EMS.

GenericCsv

Generic CSV-based collector.

Huawei U2000 iManager Collector

This collector discovers physical and logical network entities managed by a Huawei iManager U2000 EMS. Physical network entities discovered include shelves, cards, Ethernet ports, and DSL ports. Logical network entities discovered are VLANs and LAGs.

Components of the EMS integration:

EMS integration is composed of several components that assist in the collection of topology data.

The components of the EMS integration are described in Table 7.

Table 7. Components of EMS integration

Component	Description
Collector finder `ncp_df_collector`	The Collector finder reads the collector host seeds from a seed table in the `collectorFinder` database. It then queries the collectors specified in this table to get a list of devices managed by the EMS associated with each collector.

Table 7. Components of EMS integration (continued)

Component	**Description**
Collector agents	Retrieves basic and detailed information about the devices on the collector. Each agent makes use of the Collector helper to retrieve this information.
CollectorDetails agent	Retrieves basic information about the devices on the collector, including sysObjectId, sysDescr, and naming data.
CollectorInventory agent	Retrieves local neighbor, entity and associated address data for each of the devices on the collector.
CollectorLayer2 agent	Retrieves layer 2 connectivity information for the devices on the collector.
CollectorLayer3 agent	Retrieves layer 3 connectivity information for the devices on the collector.
CollectorVpn agent	Retrieves layer 2 and layer 3 VPN data for the devices on the collector.
Collector helper `ncp_dh_xmlrpc`	Enables Network Manager to communicate with the collectors using the XML-RPC interface.

Related reference:

"Topology data stored in an EMS" on page 320
There are several discovery agents that retrieve information about devices managed by an EMS.

Configuring an EMS discovery

Configure an EMS discovery to collect topology data from Element Management Systems and integrate this data into the discovered topology.

You configure an EMS discovery in the same way that you configure the discovery of any other type of network. In addition to the standard discovery configuration activities you must perform some EMS-specific discovery configuration activities.

To configure an EMS discovery, do the following activities in addition to standard discovery configuration activities:

- Configure and start the EMS collectors
- Seed the EMS discovery by seeding the Collector finder
- Enable collector discovery agents

These EMS-specific discovery configuration activities are described in the following topics.

Configuring collectors:

You can configure a collector to pass data requests and responses between Network Manager and the associated EMS or other data source.

Configuring the collector depends on the type of data source:

- For EMS: specify the hostname, port, username and password of the EMS.
- For CSV file: specify details of the CSV files and how to parse them.

You must also instruct the collector which port to listen on for XML-RPC requests from Network Manager. This is typically a one-time setup task required when a new collector is added to your Network Manager installation.

To configure a collector:

1. Edit the collector configuration file. For example, to configure the collector for the Alcatel 5620 SAM EMS, edit the file, Alcatel5620SamSoapCollector.cfg, located in:

   ```
   $NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/
   ```

2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This is also the port the collector uses to provide XML-RPC responses to Network Manager. By default this is port 8081. To make this change, find and make any changes to the General section of the configuration file, as shown in the following example:

   ```
   General =>
   {
           Debug => 0,
           Listen => 8081
   },
   ```

3. Specify the data source for this collector. This varies depending on the type of data source that the collector is using:

   - If this is a SOAP collector and the data source is an EMS, then specify the hostname and port of the EMS, together with a username and password on the EMS. To make this change, find and make any changes to the DataSource section of the configuration file, as shown in the following example:

     ```
     DataSource =>
     {
             Host => 192.168.1.2,
             Port => 8080

             Username => 'oss',
             Password => 'myPa55w0rd'
     ...
     ...
     ...
     },
     ```

   - If this is a CSV collector and the data source is a CSV file, specify the filename of the CSV file. To make this change, find and make any changes to the DataSource section of the configuration file, as shown in the following example:

     ```
     DataSource =>
     {
             CsvCfg => 'exampleCsv.cfg',
     ...
     ...
     ...
     },
     ```

4. Save the collector configuration file.

Configuring the Alcatel5620SamSoap collector:

To use data from the Alcatel5620SamSoap collector in a network discovery, you must configure the connection details between the EMS and Network Manager.

You can also configure additional information to be collected from the EMS. To configure the Alcatel5620SamSoap collector, complete the following steps:

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/Alcatel5620SamSoapCollector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

   ```
   General =>
   {
           Debug => 0,
           Listen => 8081
   },
   ```

   The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.
3. Edit the `DataSource` section of the configuration file. Specify the hostname and port of the EMS, and the username and password to connect to the EMS, as shown in the following example:

   ```
   DataSource =>
   {
           Host => 192.168.1.2,
           Port => 8080

           Username => 'oss',
           Password => 'myPa55w0rd'
   ...
   ...
   ...
   },
   ```
4. Optional: If you want to retrieve custom data from the EMS in addition to the data retrieved by default, complete the following steps.
   a. Create a configuration file in the collector directory, for example `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/extraInfo.cfg`.
   b. Edit the new file and specify the data to be retrieved, as in the following example:

      ```
                  Device =>
                  {
                      extraFields => [ { srcField => 'version', destField =>
       'm_Version', typeField => 'string' }]
                  },
      ```

      Where `srcField` is the name of the field in the SAM object, `destField` is the name of the field to which the data will be mapped within the extraInfo field, and typeField is an optional type descriptor.

      The field you want to retrieve must be part of one of the objects already retrieved by the collector. The objects queried by the collector are:

      - netw.NetworkElement

- equipment.PhysicalPort
- lag.Interface
- equipment.MediaAdaptor
- equipment.PhysicalPort
- equipment.DaughterCard
- equipment.Equipment
- equipment.Shelf
- vpls.L2AccessInterface
- vll.L2AccessInterface
- l3fwd.ServiceSite
- vprn.L3AccessInterface
- netw.PhysicalLink
- lldp.RemotePeer.

Valid types are `int` and `string`.

c. Save and close the new configuration file.

d. Edit the `CustomData` section of the `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/Alcatel5620SamSoapCollector.cfg` collector configuration file. Specify the name and location of the configuration file that defines the extra information to be collected, as in the following example:

```
CustomData =>
    {
    ExtraInfoCfg => 'extraInfo.cfg'
},
```

5. Save the collector configuration file.

Configuring the Alcatel5620SamSoapFindToFile collector:

To use data from the Alcatel5620SamSoapFindToFile collector in a network discovery, you must configure the connection details between the EMS and Network Manager, and the FTP details using which the XML files can be sent to the Network Manager server.

You can also configure additional information to be collected from the EMS. To configure the Alcatel5620SamSoapFindToFile collector, complete the following steps:

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoapFindToFile/Alcatel5620SamSoapFindToFileCollector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

```
General =>
{
        Debug => 0,
        Listen => 8081
},
```

The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.

3. Configure the following FTP parameters:

   FtpUsername
   The FTP username on the Network Manager server.

   FtpPassword
   The FTP password on the Network Manager server.

   FtpHost
   The IP address of the Network Manager server.

   PasswordFtpDirectory
   The directory on the Network Manager server where the files are to be sent.

   Tip: Copy these files to another location before the next discovery so that they do not get overwritten.

4. Edit the `DataSource` section of the configuration file. Specify the hostname and port of the EMS, and the username and password to connect to the EMS, as shown in the following example:

```
DataSource =>
{
        Host => 192.168.1.2,
        Port => 8080

        Username => 'oss',
        Password => 'myPa55w0rd'
...
...
...
},
```

5. Optional: If you want to retrieve custom data from the EMS in addition to the data retrieved by default, complete the following steps.

   a. Create a configuration file in the collector directory, for example `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/extraInfo.cfg`.

   b. Edit the new file and specify the data to be retrieved, as in the following example:

```
                Device =>
                {
                    extraFields => [ { srcField => 'version', destField =>
 'm_Version', typeField => 'string' }]
                },
```

   Where `srcField` is the name of the field in the SAM object, `destField` is the name of the field to which the data will be mapped within the extraInfo field, and typeField is an optional type descriptor.

   The field you want to retrieve must be part of one of the objects already retrieved by the collector. The objects queried by the collector are:

   - netw.NetworkElement
   - equipment.PhysicalPort
   - lag.Interface
   - equipment.MediaAdaptor
   - equipment.PhysicalPort
   - equipment.DaughterCard

- equipment.Equipment
- equipment.Shelf
- vpls.L2AccessInterface
- vll.L2AccessInterface
- l3fwd.ServiceSite
- vprn.L3AccessInterface
- netw.PhysicalLink
- lldp.RemotePeer.

Valid types are `int` and `string`.

c. Save and close the new configuration file.

d. Edit the `CustomData` section of the `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/Alcatel5620SamSoapCollector.cfg` collector configuration file. Specify the name and location of the configuration file that defines the extra information to be collected, as in the following example:

```
        CustomData =>
            {
            ExtraInfoCfg => 'extraInfo.cfg'
        },
```

6. Save the collector configuration file.

Configuring the Alcatel5620Csv collector:

To use data from the Alcatel5620Csv collector in a network discovery, you must configure the connection details between the EMS and Network Manager.

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/Alcatel5620SamCsv/Alcatel5620SamCsvCollector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

```
General =>
{
        Debug => 0,
        Listen => 8081
},
```

   The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.
3. Edit the `DataSource` section of the configuration file and specify the filename of the CSV file, as shown in the following example:

```
DataSource =>
{
        CsvCfg => 'exampleCsv.cfg',
...
...
...
},
```

4. Save the collector configuration file.

Configuring the HuaweiU2000Imanager collector:

To use data from the HuaweiU2000Imanager collector in a network discovery, you must configure the connection details between the EMS and Network Manager.

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/HuaweiU2000iManagerTL1/HuaweiU2000iManagerTL1Collector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

   ```
   General =>
   {
           Debug => 0,
           Listen => 8081
   },
   ```

   The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.
3. Edit the `DataSource` section of the configuration file. Specify the hostname and port of the EMS, and the username and password to connect to the EMS, as shown in the following example:

   ```
   DataSource =>
   {
           Host => 192.168.1.2,
           Port => 8080

           Username => 'oss',
           Password => 'myPa55w0rd'

           GetEntities => 1

              DataAcquisition =>
              {
                  StoreONTs => 1,
              }
   ...
   ...
   ,
   ```

   Set GetEntities to 1 if you want to collect entity information from the collector.

   Set StoreONTs to 1 if you want to retrieve ONT data.
4. Save the collector configuration file.

Configuring the Alcatel5529IdmSoap collector:

To use data from the Alcatel5529IdmSoap collector in a network discovery, you must configure the connection details between the EMS and Network Manager.

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/Alcatel5529IdmSoap/Alcatel5529IdmSoapCollector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

```
General =>
{
        Debug => 0,
        Listen => 8081
},
```

The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.

3. Ensure that the `Batchsize` parameter is set to 500, unless otherwise advised by IBM Support. This parameter controls the size of the SOAP/XML response.
4. Edit the `DataSource` section of the configuration file. Specify the hostname and port of the EMS, the username and password to connect to the EMS, and other options, as shown in the following example:

```
DataSource =>
{
        Host => 192.168.1.2,
        Port => 8080

        Username => 'oss',
        Password => 'myPa55w0rd'

       Domain => 'AMS'

      GetEntities => 1

       GetOnt => 0
,
```

In the above example:

- The domain of the AMS system on which the Inventory Data Manager is running is AMS.
- Collection of entity information is enabled (GetEntities => 1).
- Collection of ONT information is disabled (GetOnt => 0)

5. Save the collector configuration file.

Configuring the GenericSVC collector:

To use data from the GenericSVC collector in a network discovery, you must configure the connection details between the EMS and Network Manager.

1. Edit the collector configuration file: `NCHOME/precision/collectors/perlCollectors/GenericCsv/GenericCsv Collector.cfg`
2. Specify the port the collector must listen on for XML-RPC requests from Network Manager.

   This port is also used by the collector to provide XML-RPC responses to Network Manager. By default the port is 8081. Find and edit the `General` section of the configuration file, as shown in the following example:

```
General =>
{
        Debug => 0,
        Listen => 8081
},
```

The port must match the port you have configured in the insert into the collectorFinder.collectorRules table in the `DiscoCollectorFinderSeeds.cfg` file when seeding the collector for a first discovery.

3. Edit the `DataSource` section of the configuration file and specify the filename of the CSV file, as shown in the following example:

```
DataSource =>
{
        CsvCfg => 'exampleCsv.cfg',
...
...
...
},
```

4. Save the collector configuration file.

Starting collectors:

Before discovery starts, all the collectors must be running. You must start the collectors or make sure the collectors are running before starting a discovery that includes collectors.

You start a collector by going to the relevant collector directory and issuing a command-line interface command. Issue the following command to start a collector (note that the command is entered on one line; options are explained in the table below):

```
ncp_perl collector_script -cfg  COLLECTOR_CONFIG_FILE
[ -csvcfg CSV_COLLECTOR_CONFIG_FILE ] [ -listen PRECISION_PORT ]
[ -debug DEBUG ] [ -logdir ] [ -nologdir DIRNAME ]
[ -help ] [ -version ]
```

Table 8. Explanation of command-line options

Option	**Explanation**
collector_script	The name of the perl script that implements the collector; for example, `main.pl`.
`-cfg` *COLLECTOR_CONFIG_FILE*	Specifies the collector configuration file.
`-csvcfg` *CSV_COLLECTOR_CONFIG_FILE*	Use this optional parameter to specify the name of a CSV file to use as a data source. You can also specify this parameter within the collector configuration file. **Restriction:** This parameter is valid only when the data source is a CSV file.
`-listen` *PRECISION_PORT*	An alternative method to specify the port on which the collector must listen for requests from Network Manager. Only specify a value here if no port value has been specified in the SOAP-based collector configuration file or in the CSV-based collector configuration file.
`-debug` *DEBUG*	The level of debugging output (1-4, where 4 represents the most detailed output).
`-logdir` *DIRNAME*	Directs log messages for each process started by CTRL to `NCHOME/log/precision`.
`-nologdir` *DIRNAME*	Directs log messages for each process started by CTRL to a separate file in the specified directory.
`-help`	All Network Manager components have a special `-help` option that displays the command line options. The component is not started even if `–help` is used in conjunction with other arguments.
`-version`	All Network Manager components have a special `-version` option that displays the version number of the component. The component is not started even if `–version` is used in conjunction with other arguments.

Seeding an EMS discovery:

Seed the EMS discovery by seeding the Collector finder. This is typically a one-time setup task required when a new collector is added to your installation.

To enable Network Manager to find the collectors, you must seed the Collector finder. Seeding the Collector finder involves specifying for each collector:

- The hostname of the device on which the collector is running
- The port on that device on which the collector is listening

If a collector is running on the same host as Network Manager, then you need only specify the port.

Note: If you are rediscovering a device using the Collector finder, then specify the IP address of the device or subnet to rediscover using the Discovery Configuration GUI.

You can seed the Collector finder to perform a discovery or to perform a partial rediscovery of a single device or subnet. If you seed the Collector finder to perform a partial rediscovery, then you can also specify a single device or subnet retrieved by the collector.

You must seed the Collector finder with the host name of the device on which the collector is running, and the port on that device on which the collector is listening. If the collector is running on the same host as Network Manager, then you need to specify only the port.

Seeding the Collector for a first discovery

You seed the Collector finder for a first discovery by appending an insert into the collectorFinder.collectorRules table to the DiscoCollectorFinderSeeds.cfg configuration file. The following insert seeds the Collector finder with a host name of 172.16.25.1, and a port of 8082. This insert means that the collector is running on a host with IP address 172.16.25.0, which is different to the host on which Network Manager is running. The override number of retries for this collector is 5.

```
insert into collectorFinder.collectorRules
(
        m_Host,
        m_Port,
        m_NumRetries
)
values
(
        "172.16.25.1",
        8082,
        5
);
```

Enabling collector discovery agents:

By default, the collector discovery agents are not enabled. You must enable these agents if you are running a discovery that includes collector-based discovery.

To enable the collector agents:

1. In the Discovery Configuration GUI, select the **Full Discovery Agents** tab.
2. Select the following agents by checking the box next to the agent:
   - **CollectorDetails**
   - **CollectorInventory**
   - **CollectorLayer2**
   - **CollectorLayer3**
   - **CollectorVpn**

   Tip: You might need to scroll down the agent list to find these agents.
3. Click **Save** to save these configuration settings to the DiscoAgents.*DOMAIN_NAME*.cfg schema file, where *DOMAIN_NAME* is the name of the discovery domain, for example NCOMS.

Related tasks:

"Activating agents" on page 27
You must enable the appropriate agents for the discovery you want to perform. You can specify agents for a full discovery or for a partial discovery.

Locations and files for EMS collectors:

Perl scripts and a plain-text configuration file for each of the default collectors are held in a separate directory within the `NCHOME/precision/collectors/perlCollectors/` directory.

Experienced users can develop new collectors to enable Network Manager to interact with other EMS. Configuration and executable files for each new collector must be placed in an appropriately named directory within the `NCHOME/precision/collectors/perlCollectors/` directory.

The default collectors are listed in the following table.

Name	Directory	Configuration file
Alcatel5620SamSoap	`NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoap/`	`Alcatel5620SamSoapCollector.cfg`
Alcatel5620SamSoapFindToFile	`NCHOME/precision/collectors/perlCollectors/Alcatel5620SamSoapFindToFile/`	`Alcatel5620SamSoapFindToFileCollector.cfg`
Alcatel5620SamCsv	`NCHOME/precision/collectors/perlCollectors/Alcatel5620SamCsv/`	`Alcatel5620SamCsvCollector.cfg`
Alcatel5529IdmSoap	`NCHOME/precision/collectors/perlCollectors/Alcatel5529IdmSoap/`	`Alcatel5529IdmSoapCollector.cfg`
GenericCsv	`NCHOME/precision/collectors/perlCollectors/GenericCsv/`	`GenericCsvCollector.cfg`
Huawei U2000 iManager Collector	`NCHOME/precision/collectors/perlCollectors/HuaweiU2000iManagerTL1/`	`HuaweiU2000iManagerTL1Collector.cfg`

Configuring a context-sensitive discovery

If you have devices that you need to discover such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

In a context-sensitive discovery, information about a device is passed from the `returns` table of the Details agent to the `despatch` table of the relevant Context agent.

The Context agents use the filters in the files with an extension of .agent to determine which devices to process. This is true for all discovery agents. If the device is not of a type which supports virtual routers, that is, does not need context-sensitive processing, it is passed directly to the Associated Address agent.

Attention: Enabling a context-sensitive discovery automatically enables all the Context agents. Disabling a context-sensitive discovery automatically disables all the Context agents. Do not manually enable or disable Context agents, either through the configuration files or through the Discovery Configuration GUI.

To enable a context-sensitive discovery, append the following insert to the `DiscoConfig.cfg` file:

```
insert into disco.config
(
     m_UseContext
)
values
(
     1
)
```

Inserting the value `0` disables the context-sensitive discovery.

Related concepts:

"Discovering device details (context-sensitive)" on page 288
The discovery of context-sensitive device details is carried out in several steps.

Related reference:

"Context-sensitive discovery agents" on page 329
There are several agents that take part in a context-sensitive discovery.

Configuring MPLS discoveries

Configure an MPLS discovery to discover core MPLS networks and the VPNs that use these core networks. Advanced MPLS discovery configuration provides extra customization facilities.

About MPLS discovery

Administrators within service providers who offer Multiprotocol Label Switching (MPLS) VPN services can discover MPLS core networks and MPLS VPNs to enable NOCs within the service provider to monitor the health of customer VPNs.

Network Manager supports the discovery of the following VPNs running across MPLS core networks:

- Layer 3 VPNs
- Enhanced Layer 2 VPNs

For the enhanced Layer 2 VPNs, Network Manager discovers point-to-point pseudowires linking two provider edge (PE) routers.

The following sections specify the terminology and topology visualization conventions used in Network Manager to refer to MPLS networks.

Note: The graphics shown in this section are conceptual representations of an MPLS network only. You cannot see these conceptual views in the Network Views graphical user interface (GUI).

Layer 3 MPLS VPNs:

Network Manager can visualize layer 3 MPLS VPN topologies in a core view or an edge view.

The core view and edge view differ as follows:

- The core view shows the provider-edge (PE) routers, and provides visibility of the provider core (P) routers and label switched path (LSP) data within the MPLS core, for each of the VPNs running across the MPLS core.
- The edge view shows the PE routers and the MPLS cloud only. It does not provide visibility of the devices in the core.

Enhanced Layer 2 MPLS VPNs:

For enhanced Layer 2 VPNs, Network Manager provides only an edge view of your MPLS core network.

Network Manager displays an enhanced Layer 2 VPN as a collection of point-to-point pseudowires. This means that if an enhanced Layer 2 VPN contains more than two provider edge (PE) routers, then Network Manager displays that VPN in multiple views, each view consisting of a single PE to PE point-to-point connection.

Table 9 shows examples of enhanced Layer 2 VPNs with two and more than two PEs. The table also provides the number of pseudowires, and hence the number of views that Network Manager displays for each VPN.

Table 9. Number of pseudowires for an enhanced Layer 2 VPN

Number of PEs in an Enhanced Layer 2 VPN	Number of point-to-point pseudowires	Number of Views Network Manager displays for this VPN
2	1	1
3	3	3
4	6	6

Standard and advanced MPLS discovery configuration:

Configure a standard MPLS discovery to discover all of your MPLS network and uses default naming convention for the VPNs discovered. The standard MPLS discovery configuration also enables the display of service-affected events (SAEs) in the **Active Event List (AEL)**. Advanced MPLS discovery configuration provides extra customization facilities.

Configuration activities for an MPLS network include seeding, scoping, and the other standard discovery activities.

Standard and advanced MPLS discovery configuration differ as follows:

- Standard MPLS discovery: discovers all of your MPLS network and uses default naming convention for the VPNs discovered
- Advanced MPLS discovery: using the advanced configuration options you can:
  - Restrict the scope of the discovery to a particular VPN or VRF
  - Configure your own VPN naming conventions
  - Force label discovery even if you selected an RT-based discovery

After you have configured and run an MPLS discovery, your operators can monitor customer VPNs in the following ways:

- View topology maps of selected VPNs, showing the alert status of the VPNs and of the devices in the VPNs .
- Identify service-affected events (SAEs) in the **Active Event List (AEL)**. An SAE is an alert that warns operators that a critical customer service, for example, a customer VPN, has been affected by one or more network events. The underlying network events are on an interface on either a PE router or a CE router.

About Service Affected Events:

A Service Affected Event (SAE) alert warns operators that a critical customer service has been affected by one or more network events.

An SAE is produced when one or more events occur on a Provider Edge (PE) or Customer Edge (CE) interface in a Virtual Private Network (VPN) or Virtual Private LAN Service (VPLS). The underlying network events are on an interface of a PE router or a CE router, or on the link between them. You must configure the MPLS discovery to infer the existence of CE routers so all possible SAEs are generated for your customer VPNs.

The following list gives two examples of SAE events generated on two different customer VPNs:

- SAE generated on customer-1 VPN because of an `Mpls VRF Down` trap on a PE router interface
- SAE generated on customer-3 VPN because of a `LinkDown` trap on a CE router interface

Each SAE appears as an alert in the Active Event List (AEL). The appearance of the SAE warns operators that a customer VPN has been affected, possibly critically, by one or more network events. Operators can right-click the SAE and issue a command to view the underlying events that caused the SAE.

For more information about the AEL, see the *IBM Tivoli Netcool/OMNIbus Web GUI Administration and User's Guide*.

Configuring standard MPLS discovery

Configure an MPLS discovery to discover core MPLS networks and the VPNs that use these core networks.

In addition to the standard discovery configuration activities, you must perform some MPLS-specific discovery configuration activities:

- Configure MPLS agents
- Specify the discovery methods, that is, whether to run a route-target or label-switched path (LSP) discovery
- Configure SNP and Telnet to ensure that the agents can access network devices
- Configure Network Manager to infer the existence of CE routers. This step is necessary to enable operators to view service-affected events in the **Active Event List (AEL)**.

These EMS-specific discovery configuration activities are described in the following topics.

Configuring MPLS agents:

As part of MPLS discovery configuration you must enable one or more MPLS agents. You can also resolve the problem of duplicate IP addresses in different Virtual Private Networks (VPNs) by configuring the AsAgent agent.

The following MPLS agents and the corresponding agent definition (.agnt) files are provided:

- Juniper Telnet agent (JuniperMPLSTelnet.agnt)
- Juniper ERX router agent (UnisphereMPLSTelnet.agnt)
- Cisco MPLS Telnet agent (CiscoMPLSTelnet.agnt)
- Cisco MPLS SNMP agent (CiscoMPLSSnmp.agnt)
- Laurel MPLS Telnet agent (LaurelMPLSTelnet.agnt)

  Note: The Laurel MPLS Telnet agent is intended for RT- (RouteTarget) based discoveries only.

These agents can discover MPLS VPN and Virtual Private LAN Service (VPLS) data from devices in the network.

Tip: Agents that retrieve VPLS information can retrieve large amounts of data. Enabling these agents can add significant processing time to the discovery process. If you do not need to rediscover VPLS information, disable these agents for a faster discovery.

Note: If you have an MPLS network that supports both Layer 3 and enhanced Layer 2 VPNs, then the same MPLS agents discover both types of VPN. Network Views can also partition both Layer 3 and enhanced Layer 2 VPNs simultaneously on the same core MPLS network.

If your MPLS network contains Cisco equipment, enable both the Cisco MPLS Telnet and Cisco MPLS SNMP agents. These two agents complement each other, as follows:

- Cisco MPLS SNMP agent targets only devices with an Internetwork Operating System (IOS) that fully supports SNMP-based MPLS discovery
- CiscoMPLSTelnet agent targets only devices running an IOS that does not fully support an SNMP-based discovery

Attention: Use caution when altering the CiscMPLSSnmp.agnt file. Some network devices might contain IOS versions that have a flaw that might affect the device when certain MPLS SNMP data is requested. These IOS versions have been filtered out by default in the CiscMPLSSnmp.agnt file.

In addition to these standard discovery configuration activities, you can also change the scope of the MPLS discovery by restricting the scope to specific VPNs or VRFs.

Related tasks:

"Defining the scope of an MPLS/VPN discovery" on page 113
When configuring the discovery of one or more Virtual Private Networks (VPNs) running across an MPLS core, you can restrict the scope of this discovery to a particular VPN name or VPN Routing and Forwarding (VRF) table name.

Configuring MPLS Telnet agents:

The CiscoMPLSTelnet, JuniperMPLSTelnet, LaurelMPLSTelnet, and UnisphereMPLSTelnet agents obtain data from devices primarily through Telnet. You must enable these agents and configure Telnet access to ensure that MPLS Telnet agents can access devices and can understand the output from devices.

Do the following to configure Telnet access for MPLS Telnet agents:

1. Populate the Telnet configuration file `TelnetStackPasswords.cfg` so the agents can access the target devices.
2. Configure the Telnet Helper so that agents can understand the output from devices.

Related tasks:

"Configuring device access" on page 23
Specify SNMP community strings and Telnet access information to enable helpers and Network Manager polling to access devices on your network.

Related reference:

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

Configuring MPLS SNMP agents:

The CiscoMPLSSnmp agent obtains data from devices using SNMP. You must enable this agent and configure SNMP access to ensure that this agent can access devices and can understand the output from devices.

To configure SNMP access for MPLS SNMP agents:

Note: The CiscoMPLSSnmp.agnt attempts to retrieve the L2 VPNs using the telnet 'show' commands if the agent fails to retrieve the data via SNMP.

1. Configure SNMP access to devices.
2. Configure the SNMP Helper so that agents can understand the output from devices.

Related tasks:

"Configuring device access" on page 23
Specify SNMP community strings and Telnet access information to enable helpers and Network Manager polling to access devices on your network.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

Configuring the AsAgent agent:

To resolve the problem of duplicate IP addresses in different VPNs, activate the AsAgent agent and provide Network Manager with a mapping file, `ASMap.txt`, that contains a complete list of devices in each VPN, together with an `AddressSpace` tag, which defines which VPN the device belongs to.

During an MPLS discovery, Network Manager might discover devices in different VPNs with identical IP addresses. In this case, Network Manager cannot differentiate between these devices and might resolve device connectivity incorrectly. The devices in question might be the CE routers at the edge of the VPNs, or might be devices within the VPNs.

In the `ASMap.txt` mapping file, provide a complete list of devices in each VPN, together with an `AddressSpace` tag, which defines which VPN the device belongs to.

Table 10 provides a description of the AsAgent agent that you need to activate to resolve the problem of duplicate IP addresses.

Table 10. AsAgent agent

Agent name	Function
AsAgent	Enables Network Manager to uniquely identify devices in different VPNs with identical IP addresses, and thereby correctly resolve device connectivity. This agent works in conjunction with the `ASRetprocessing.stch` stitcher and with the `ASMap.txt` file in `NCHOME/precision/etc`.

Table 11 provides the format of the `ASMap.txt` file by showing an example of the content of this file. Fields in this text file must be separated by tabs.

Table 11. Format of ASMap.txt file

Base Name	Address Space	IP Address
`CERouter-1`	`CUSTOMER-1`	`192.168.2.1`
`CEDevice-a`	`CUSTOMER-1`	`192.168.2.21`
`CEDevice-b`	`CUSTOMER-1`	`192.168.2.22`
`CEDevice-c`	`CUSTOMER-1`	`192.168.2.23`
`CERouter-2`	`CUSTOMER-2`	`192.168.2.1`
`CEDevice-a`	`CUSTOMER-2`	`192.168.2.31`
`CEDevice-b`	`CUSTOMER-2`	`192.168.2.32`

Configuring MPLS discovery method:

You can configure MPLS discovery in either of two ways: Route Target (RT)-based discovery; Label Switched Path (LSP)-based discovery.

The methods to configure MPLS discovery are:

- Route Target (RT)-based discovery: Network Manager uses VRF and RT information to determine which provider edge routers are involved in a VPN.
- Label Switched Path (LSP)-based discovery: Network Manager uses VRF and LSP information to determine which provider edge (PE) routers are involved in a VPN and which provider core (P) routers are traversed by the LSPs within that VPN.

Choose which MPLS discovery method to use by setting the **Enable RT Based MPLS VPN Discovery** check box in the Discovery Configuration GUI.

- Check the **Enable RT Based MPLS VPN Discovery** check box to enable RT-based MPLS discovery.
- Clear the **Enable RT Based MPLS VPN Discovery** check box to enable LSP-based MPLS discovery.

You can also perform this configuration manually by setting the value of the field `m_RTBasedVPNs` in the `disco.config` table.

Note: RT-based discoveries are based on a more later technology than LSP-based discoveries, and can improve discovery performance. To avoid performance problems with LSP-based MPLS/VPN discoveries, use the default RT-based option. The RT-based option is the default option set on the **Advanced** tab of the **Network Discovery Configuration** page. You can use VRF names with the RT-based option by editing the configuration file as described in “Using VRF names with RT-based discoveries” on page 109.

Table 12 summarizes the differences between RT-based discovery and LSP-based discovery.

Table 12. RT-based discovery and LSP-based discovery

Type of Discovery	Label Data	Core View	VPN Resolution
RT-based discovery	No label data is required for this type of discovery Discovery is faster	Consists of all MPLS-enabled devices	VPNs resolved based on RT information
LSP-based discovery	Label data is discovered in order to trace LSPs Discovery is slower	Consists of devices traversed by the relevant LSPs	VPNs resolved based on VRF and label path information

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

"disco.config table" on page 183
The config table configures the general operation of the discovery process.

Using VRF names with RT-based discoveries:

You might prefer the LSP-based discoveries in order to use the more familiar VRF name for the VPNs. However, you can also use VRF names with the RT-based discoveries.

To use VRF names with RT-based discoveries:

1. Exit all instances of the **Discovery Configuration** GUI.
2. Go to the `NCHOME/etc/precision` directory.
3. Edit the `DiscoConfig.`*`DomainName`*`.cfg` file as follows:
   a. Set the **`m_RTVPNResolution`** field to `2` in the disco.config table.
   b. Ensure that the **`m_RTBasedVPNs`** value is set to 1.
4. Restart the ncp processes to read the configuration files again:

   ```
   itnm_stop ncp
   itnm_start ncp
   ```

   Alternatively, restart the ncp_config process.

Inferring the existence of CE routers:

You can infer the existence of your customers' CE routers by making specifications in the advanced discovery configuration options within the Discovery Configuration GUI.

If the host on which Network Manager is installed has no access to your customers' CE routers, then Network Manager cannot discover these routers directly. This situation typically occurs when the company providing MPLS services owns the PE routers but has no access to CE routers, which are owned by the customers running the VPNs.

Note: This situation does not occur if the company providing MPLS services owns and manages both PE and CE routers and therefore has access to both sets of devices.

To infer the existence of your customers' CE routers, specify this in the advanced discovery configuration options within the Discovery Configuration GUI.

Note: You should do this only where the PE interface is on a /30 subnet. In this case, the other device on the subnet must be the CE router, and the IP address of the CE will be the other address on the /30 subnet.

Limitations on inferring CE routers:

- Avoid inferring the existence of CE routers if your PE routers are connected to the CE routers by serial links and you know that there is IP address duplication among the CE routers and devices within the MPLS core network. Network

Manager will remove from the topology any discovered MPLS core routers that have the same IP address as an inferred CE IP address.

- If your PE routers are connected to the CE routers by Ethernet, you can infer the existence of CE routers without needing to perform any other checks. In this case, Network Manager can determine the MAC address of the CE router. If Network Manager has discovered another device with the same MAC address, then it must be the CE router. In this case, Network Manager uses the discovered device data and does not infer the existence of the CE.

Related reference:

"Advanced discovery parameters" on page 37
Advanced settings control features of the discovery such as concurrent processes and timeouts. Use these parameters to increase the speed of the discovery, but balance the speed with the load on the server. Generally, a faster discovery results in more memory usage on the server.

Configuring advanced MPLS discovery

Configure an advanced MPLS discovery for extra customization facilities not included in the standard MPLS discovery.

When you configure an advanced MPLS discovery, you must perform the following activities in addition to the activities required for a standard MPLS discovery.

- Define the scope of the MPLS discovery: enables you to restrict the scope of this discovery to a particular VPN or VRF.
- Specify a VPN name: enables you to configure your own VPN naming convention
- Fine-tune label data discovery: enables you to force LSP discovery regardless of the MPLS discovery method selected

Configuring discovery of MPLS Traffic Engineered tunnels:

To discover MPLS Traffic Engineered tunnels, enable the StandardMPLSTE agent, configure the information that is retrieved, and configure the scope of the discovery.

MPLS Traffic Engineered tunnel discovery modes:

Set the discovery mode according to how much detail you want to retrieve.

A mode-switch is provided in the discovery agent configuration file that configures specific tunnel instances, which can be wildcarded, to retrieve different amounts of tunnel data. You can choose any of the following modes.

HeadEndHops (default)

In HeadEndHops mode, the agent retrieves head-end and tail-end of the tunnel, and the transit LSRs and next-hop interfaces are identified by querying the head-end LSR for computed and actual-route hop data. The actual and computed route data is retrieved from the mplsTunnelARHopTable and mplsTunnelCHopTable MIB tables respectively. This discovery mode does not store transit and tail-end tunnel instances against transit and tail-end LSRs. A connection is created in the MPLS TE topology between the head-end and tail-end LSR interfaces via transit device hops (if present) which are associated with the head-end LSR tunnel object for the appropriate tunnel interface.

MPLS cross-connect pointers that are discovered and resolved on the head tunnel will be resolved to the appropriate LSP ID where possible.

You can use this information to determine if the actual path taken by a tunnel is different to the path computed by the Compute Shortest Path First (CSPF) calculations. You can see the computed and actual path, although there is no way to determine that an LSR is acting in a transit or tail capacity without looking at the head-end LSR tunnel data.

Note: Actual route data is only available if the Record Route Option (RRO) has been specified for the tunnel instance.

In the schema of the scope.mplsTe table, the HeadEndHops mode maps to value 1 of m_Mode.

HeadTailEnd

In HeadTailEnd mode, only MPLS TE tunnel head-end and tail-end points are resolved, by querying the head-end Label Switching Router (LSR). This mode provides the minimal amount of information about the MPLS TE tunnels. A connection in the MPLS TE topology is created between the head-end and tail-end LSR interfaces. A tunnel resource instance is associated with the head-end tunnel LSR entity.

In this mode, you cannot identify the transit LSRs, and computed and actual route data is not retrieved.

MPLS cross-connect pointers that are discovered and resolved on the head tunnel will be resolved to the appropriate LSP ID where possible.

In the schema of the scope.mplsTe table, the HeadTailEnd mode maps to value 2 of m_Mode.

AllLSRTunnelsAndHops

In AllLSRTunnelsAndHops mode, the agent retrieves the head-end and tail-end of the tunnel and identifies transit LSRs and next-hop interfaces by querying the head-end LSR for computed and actual route hop data. The actual and computed route data is retrieved from the mplsTunnelARHopTable and mplsTunnelCHopTable MIB tables respectively. This discovery mode stores transit and tail-end tunnel instances against transit and tail-end LSRs. The mode creates a connection in the MPLS TE topology between the head-end and tail-end LSR interfaces that are associated with the head-end (for the tunnel interface) and transit and tail-end LSR tunnel objects. Computed and actual-route connections are associated with Computed and Actual connection entity types, which are aggregated in sequence from the head-end LSR tunnel entity. A tunnel resource instance is associated with the head-end tunnel LSR entity.

You can use this information to determine if the actual path taken by a tunnel is different to the path computed by the CSPF calculations. You can see the computed and actual path and determine the transit or tail-end role of an LSR without looking at the headend LSR tunnel instance.

Note: Actual route data is only available if the Record Route Option (RRO) has been specified for the tunnel instance.

MPLS cross-connect pointers that are discovered and resolved on the head tunnel will be resolved to the appropriate LSP ID where possible.

In the schema of the scope.mplsTe table, the AllLSRTunnelsAndHops mode maps to value 3 of m_Mode.

Related reference:

"mplsTe table" on page 203
The mplsTe table defines the scope of MPLS Traffic Engineered (TE) tunnel discovery, and defines what information is retrieved.

Enabling the StandardMPLSTE agent:

To discover MPLS TE tunnels, you must enable the StandardMPLSTE agent and add the relevant SNMP community strings.

To enable the StandardMPLSTE agent, complete the following steps.

1. Click **Discovery** > **Network Discovery Configuration**. From the **Domain** list, select the required domain.
2. Click the **Full Discovery Agents** tab. The Agents List is displayed, showing all available discovery agents for the selected discovery option.
3. Select the check box next to the StandardMPLSTE agent.
4. Click **Save** 💾.
5. Optional: If you want to rediscover MPLS TE tunnels, enable the StandardMPLSTE agent for partial rediscoveries.
   a. Click the **Partial Rediscovery Agents** tab.
   b. Select the check box next to the StandardMPLSTE agent.
   c. Click **Save** 💾.
6. Ensure that the SNMP community strings are configured correctly to access the devices in the MPLS TE tunnels.

Related tasks:

"Configuring device access" on page 23
Specify SNMP community strings and Telnet access information to enable helpers and Network Manager polling to access devices on your network.

Configuring the StandardMPLSTE agent:

Configure which tunnels to discover, and what details to retrieve.

To configure the StandardMPLSTE agent, complete the following steps.

1. Back up and edit the file `NCHOME/etc/precision/DiscoScope.cfg`.
2. Locate and edit the insert into the scope.mplsTe table, or create a new insert. Create or edit an insert similar to the following:

```
insert into scope.mplsTe
(
   m_Protocol,
   m_Zones,
   m_Mode,
   m_TunnelFilter
)
values
(
   1,
    [{m_Subnet = '192.168.1.0', m_NetMask = 24 }],
    2,
    1
);
```

This insert configures the agent to behave in the following way:

- It uses IPv4.
- It includes (m_Tunnelfilter=1) the subnet 192.168.1.* in the discovery of tunnel heads.
- It retrieves data for the head and tail of the tunnel but not for the transit routers.

3. Save and close the file.
4. Stop and restart the discovery engine, the **ncp_disco** process, for your configuration changes to take effect.

Related reference:

"mplsTe table" on page 203
The mplsTe table defines the scope of MPLS Traffic Engineered (TE) tunnel discovery, and defines what information is retrieved.

Defining the scope of an MPLS/VPN discovery:

When configuring the discovery of one or more Virtual Private Networks (VPNs) running across an MPLS core, you can restrict the scope of this discovery to a particular VPN name or VPN Routing and Forwarding (VRF) table name.

You restrict the scope by configuring the optional DiscoAgentDiscoveryScoping section in the *.agnt file. The configurable options are described in Table 13.

Table 13. Defining MPLS scoping requirements

Option	Function
IncludeVRF	Allows the discovery of the named VRF
IncludeVPN	Allows the discovery of the named VPN
ExcludeVPN	Does not discover any VRFs within the named VPN
ExcludeVRF	Does not discover the specified VRF

The order of precedence for Exclude and Include within the DiscoAgentDiscoveryScoping section is:

1. Exclude
2. Include

The order of precedence for VRF and VPN within the DiscoAgentDiscoveryScoping is:

1. VRF
2. VPN

For example, if you include a VPN, but another filter excludes a VRF in your VPN, then the VRF is excluded. If a VPN is excluded, but another filter includes a VRF within that VPN, then the VRF is included.

VRF names are case sensitive and an asterisk (*) represents a wildcard for any VRF or VPN name when used in the name part of the configuration. The wildcard can be used with any of the above options.

Scoping by VPN name works only when the VRF names configured on the devices discovered by the MPLS agents are in the Cisco-recommended VRF format. A VRF is named based on the VPN or VPNs serviced, and the topology type. The format for the VRF names are:

```
V [number assigned to make the VRF name unique]: [VPN_name]
```

For example, in a VPN called `precision`, a VRF for a hub edge router would be:

```
V1:precision
```

A VRF for a spoke edge router in the `precision` VPN would be:

```
V1:precision-s
```

A VRF for an extranet VPN topology in the `precision` VPN would be:

```
V1:precision-etc
```

The following example scopes a discovery in a system where there are four VRFs: `V65:Precision-etc`, `V65:Precision-s`, `V65:Precision`, and `V44:AcmeSheds`.

```
//2 VRFs are to be included
//
DiscoAgentDiscoveryScoping
{
    IncludeVRF = "V65:Precision-etc";
    IncludeVRF = "V44:AcmeSheds";
}
//All 4 VRFs are to be included
//
DiscoAgentDiscoveryScoping
{
    IncludeVPN = "Precision";
    IncludeVRF = "V44:AcmeSheds";
}
```

Related reference:

"disco.config table" on page 183
The config table configures the general operation of the discovery process.

Configuring VPN naming conventions:

If you do not use the Cisco VRF naming convention, you can configure your own VPN naming convention by making the appropriate inserts into the `MPLSAddVPNNames.stch` stitcher located in `$NCHOME/precision/disco/stitchers/`.

The `MPLSAddVPNNames` stitcher extracts and constructs a VPN name from the list of paths discovered by the Path Tracing stitchers. The `MPLSAddVPNNames` stitcher can then add the VPN name to the device interfaces that fall under the paths belonging to the VPN.

The following example shows where to modify the VPN name in the `MPLSAddVPNNames.stch` file, located in `$NCHOME/precision/disco/stitchers.`

```
//VPN Name Assignment
//
//Currently assigns the VRF name as the VPN Name if no VPN name
//has been discovered by the agent, i.e., if the VRF name was not in
//the Cisco format.
//
vpnName = eval(text, '&m_VPNName');
```

```
if (vpnName == NULL)
{
    vpnName = vrfName;    //VPN=VRF, customize as required.
}
```

Fine-tuning label data:

The MPLS discovery method (RT-based or LSP-based) determines whether the MPLS agents retrieve MPLS label data.

- If you choose RT-based discovery, the MPLS agents do not retrieve label data.
- If you choose LSP-based discovery, the MPLS agents retrieve label data.

If you choose an RT-based discovery, but want to retrieve label data, it is possible to do this manually with the following insert in the DiscoAgentDiscoveryScoping section of the appropriate MPLS.agnt file:

```
DiscoAgentDiscoveryScoping
{
    GetMPLSLabelData = 1;
}
```

Related tasks:

"Configuring MPLS discovery method" on page 108
You can configure MPLS discovery in either of two ways: Route Target (RT)-based discovery; Label Switched Path (LSP)-based discovery.

Configuring NAT discoveries

Configure a Network Address Translation (NAT) discovery to discover NAT environments, by mapping the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

About Network Address Translation

The number of available IP addresses in the current 32–bit format is not enough to meet the growth in demand for access to the Internet. Network Address Translation (NAT) was designed as a short-term solution to this problem by providing a method of connecting multiple computers to an IP network using either a single unique public IP address, or a small number of unique public IP addresses.

NAT is commonly used in corporations, where a NAT router sits at the edge of the private network (referred to in this context as a *stub* domain) and translates the IP addresses attached to packets entering and leaving the stub domain. The NAT router, which effectively acts as an agent between the Internet and the local network, maintains a list of the mappings between public and private addresses.

Note: A stub domain is a local network using internal IP addresses. The network can use unregistered, private, IP addresses for internal communication—these addresses must be translated into unique, public, IP addresses when communicating outside the network. The addresses used internally by a given stub domain can also be used internally by any other stub domain.

For example, when a computer within the private network requests information from the public network, the NAT router automatically translates the private address of that computer into the public address of the domain, which is the only address that is transmitted to the public network. When the requested information is returned, the NAT router consults its internal list of public to private address mappings in order to forward the information to the appropriate computer.

There are a number of different ways to configure a NAT environment. The following descriptions detail the most common types of NAT environment.

Static NAT Environments:

In a static NAT environment, the NAT router maps private and public addresses on a one-to-one basis, that is, the private address of a given device always maps to the same public address. This type of NAT environment is commonly used for devices that need to be accessible to the public network.

Dynamic NAT environments:

In a dynamic NAT environment, the NAT router dynamically allocates public IP addresses, from a group of addresses, to devices on the private network that wish to communicate with the public network. A variation on dynamic NAT, *overloading* or PAT (Port Address Translation), maps multiple private addresses to the same public address using different ports.

Private Address Ranges:

The Internet Assigned Numbers Authority (IANA) has assigned several address ranges to be used by private networks.

Address ranges to be use by private networks are:

- Class A: `10.0.0.0` to `10.255.255.255`
- Class B: `172.16.0.0` to `172.31.255.255`
- Class C: `192.168.0.0` to `192.168.255.255`

An IP address within these ranges is therefore considered non-routable, as it is not unique. Any private network that needs to use IP addresses internally can use any address within these ranges without any coordination with IANA or an Internet registry. Addresses within this private address space are only unique within a given private network.

All addresses outside these ranges are considered public.

About NAT discovery

You can use Network Manager to manage NAT environments, though there are some restrictions on the types of NAT environment that are currently supported.

Network Manager can interrogate known, supported NAT gateways to obtain a list of public to private IP address mappings of devices in NAT domains. Alternatively, these mappings can be supplied manually. Network Manager can then discover those devices behind the NAT gateways that have a public IP address.

Each NAT domain has a unique address-space identifier. Each device in a NAT domain has the appropriate address-space identifier appended to its record. This enables the devices to be managed (for example, polled).

Restrictions on NAT discovery:

There are several restrictions on the management of NAT environments using Network Manager.

Management of NAT environments using Network Manager is restricted by the following conditions:

- Network Manager can discover one or more NAT environments, but they must all use static NAT address mapping.
- Network Manager can discover devices in multiple NAT domains, regardless of whether the private IP addresses of the devices are duplicated in other NAT domains. However, the public IP address of each device in each domain must be unique.
- Devices within a NAT domain that have only private IP addresses cannot be discovered or managed by Network Manager.
- The discovery process must discover the NAT environment from outside, that is, from the public network.
- Virtual IP addresses such as Hot Standby Routing Protocol (HSRP) addresses cannot be mapped. The real physical address must be used.
- The following must be supplied before the discovery is run:
  - The addresses of all supported NAT gateways.
  - The NAT gateway translations must be discovered, either automatically or by supplying the NATTextFileAgent discovery agent with a flat file of public-to-private IP address mappings.

Differences in a NAT discovery process flow:

The process flow of a NAT discovery differs from the process flow of a normal discovery.

Related concepts:

"Discovery cycles" on page 285
A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

Downloading translation information:

NAT translation information is downloaded by the NAT agents into the translations.NATTemp database table before the finders process any other entities.

All other discovered devices are inserted into the finders.pending table while the BuildNATTranslation.stch stitcher creates a global translation table and stores it in the translations.NAT database table.

The finders, helpers, and other components that need to access devices can use this table to look up the address of any device behind a NAT gateway.

Creating the topology:

When the topology is created, the `AddBaseNATTags.stch` stitcher adds NAT information to the topology record of each device in a NAT domain.

Table 14 shows the information that is added to the topology record for each device.

Table 14. NAT information added to a device record

Column	Description
`ExtraInfo->m_AddressSpace`	The name of the NAT address space to which the device belongs. This value is set in the `translations.NATAddressSpaceIds` table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
`ExtraInfo->m_NATTranslated`	A Boolean integer indicating whether the device lies behind a NAT gateway.
`ExtraInfo->m_InsideLocalAddress`	The private address of the device.
`ExtraInfo->m_OutsideGlobalAddress`	The public address of the device.

Configuring a NAT discovery

Configure a NAT discovery to discover NAT environments and to enable Network Manager to manage NAT environments.

You set most of the NAT discovery settings from the Discovery Configuration GUI, except for the following tasks:

- Configure the NATTextFileAgent agent to provides support for any unsupported NAT gateway devices
- Configure the NATGateway agent to correct the potential problem of incorrect connectivity when the NAT gateway is not in the public address space.

Quick reference for NAT discovery configuration:

Use this information as a step-by-step guide to configuring a NAT discovery..

The steps are described in the following table.

Table 15. Quick reference for NAT discovery configuration

Action	Using the GUI	Using the command line
1. Configure the discovery to use network address translation. You can do this using the Discovery Configuration GUI, or using the command line.	“Configuring NAT translation” on page 33	“Enabling NAT translation” on page 120
2. Define each NAT gateway device and its corresponding address space. You can do this using the Discovery Configuration GUI, or using the command line.		“Defining address spaces for NAT gateways” on page 120

Table 15. Quick reference for NAT discovery configuration (continued)

Action	Using the GUI	Using the command line
3. Seed the Ping finder with the IP address of each NAT gateway device.	"Seeding discovery" on page 20	Guidance for seeding a discovery "DiscoPingFinderSeeds.cfg configuration file" on page 61 Guidance for seeding a NAT discovery "Seeding discovery with NAT gateway addresses" on page 122
4. Define a scope zone for each NAT gateway device. **Note:** You do not need to define a scope zone for any NAT Gateway devices whose IP address is already within any other scope zones defined for the discovery. **Note:** Do not define an address space for the NAT gateway devices or for public subnet scopes. Address space can only be defined for private subnets.	"Scoping discovery" on page 17	Guidance for scoping a discovery "DiscoScope.cfg configuration file" on page 64 Example: how to define a scope zone for a private NAT subnet "Defining a scope zone within a NAT domain" on page 121
5. Define scope zones for the public subnets associated with each NAT address space. **Note:** Do not define an address space for the NAT gateway devices or for public subnet scopes. Address space can only be defined for private subnets.		
6. Where possible, define scope zones for the private subnet associated with each NAT address space. **Restriction:** You can only define a scope zone for a private NAT address space where the subnet and netmask combination of the private subnet is unique within the discovery configuration. Make the following settings when defining this scope: 1. Uncheck the **Add to Ping Seed List** option. You must do this because private subnets are not pingable. 2. Define an address space for this private subnet. The advantages of adding a scope zone for each private NAT address space are as follows: • This ensures that only addresses in that private space are fed back during the discovery. • If the NAT Gateway device and the devices within the associated NAT address space are routers. then adding a scope zone for that private NAT address space limits the download of unnecessary routing data.		
7. Enable NAT agents as follows: • For supported NAT Gateway devices, enable the CiscoNATTelnet or NATNetScreen agent. • For unsupported NAT Gateway devices, create a NAT mapping file and enable the NATTextFileAgent agent	"Activating agents" on page 27	"Enabling agents for supported NAT gateway devices" on page 123 "Enabling agents for unsupported NAT gateway devices" on page 123

Related tasks:

"Example: Configuring a NAT discovery" on page 125
This example illustrates how to define address spaces using the NATTextFileAgent agent and how to set up associated discovery scopes.

Enabling NAT translation:

You can set the discovery system to use NAT translation by editing $NCHOME/etc/precision/DiscoConfig.cfg to create or amend an insert into disco.NATStatus to set m_UsingNAT to 1 and m_NATStatus to 0.

The completed insert must resemble the following:

```
insert into disco.NATStatus
(
        m_UsingNAT,
        m_NATStatus
)
values
(
        1,
        0
);
```

Related tasks:

"Configuring NAT translation" on page 33
To configure NAT translation to discover NAT environments, map the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

"Enabling NAT translation"
You can set the discovery system to use NAT translation by editing $NCHOME/etc/precision/DiscoConfig.cfg to create or amend an insert into disco.NATStatus to set m_UsingNAT to 1 and m_NATStatus to 0.

Defining address spaces for NAT gateways:

To specify the IP address of your NAT gateways and the address space identifier you want to use for each associated NAT domain, edit `DiscoConfig.cfg` to create or amend an insert into `translations.NATAddressSpaceIds`.

Follow these guidelines when defining address spaces for NAT gateways:

- The IP address must be the public IP address that is accessible from the management server.
- The address space field can be any descriptive string, but avoid special characters such as quotes. Use the standard rules for DNS names for the address space because the address space might make up part of the name of these devices.

The following example insert configures the discovery system for two NAT gateways.

```
insert into translations.NATAddressSpaceIds
(
        m_NATGatewayIP,
        m_AddressSpaceId
)
values
(
         '172.16.1.112',
        'NATDomain1'
);

insert into translations.NATAddressSpaceIds
(
        m_NATGatewayIP,
        m_AddressSpaceId
)
values
(
         '172.16.1.104',
        'NATDomain2'
);
```

Related tasks:

"Configuring NAT translation" on page 33
To configure NAT translation to discover NAT environments, map the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

"Enabling NAT translation" on page 120
You can set the discovery system to use NAT translation by editing $NCHOME/etc/precision/DiscoConfig.cfg to create or amend an insert into disco.NATStatus to set m_UsingNAT to 1 and m_NATStatus to 0.

Defining a scope zone within a NAT domain:

You can customize inclusion and exclusion zones for individual NAT domains, using the `m_AddressSpace` column of the `scope.zones` table.

The following example insert defines an inclusion zone for a private subnet associated with a NAT domain.

```
insert into scope.zones
(
    m_Protocol, m_Action, m_Zones, m_AddressSpace
)
values
(
        1,
        1,
        [
            {
                m_Subnet="172.16.2.*",
            }
        ],
        "NATDomain1"
);
```

The above example defines one inclusion zone. Network Manager discovers any device with an IP address starting with `"172.16.2"`, that is, in the private `172.16.2.0` subnet with a mask of `255.255.255.0`, and that also belongs to the NAT address space `NATDomain1`. The protocol is set to 1, that is, IP.

Note: Do not define an address space for the NAT gateway devices or for public subnet scopes. Address space can only be defined for private subnets.

Related tasks:

"Configuring NAT translation" on page 33
To configure NAT translation to discover NAT environments, map the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

"Enabling NAT translation" on page 120
You can set the discovery system to use NAT translation by editing $NCHOME/etc/precision/DiscoConfig.cfg to create or amend an insert into disco.NATStatus to set m_UsingNAT to 1 and m_NATStatus to 0.

Seeding discovery with NAT gateway addresses:

Seed a NAT discovery by inserting into the Ping finder the IP addresses of the main routers within the system. Also seed the discovery with the IP addresses of the NAT gateway IPs.

In a NAT-based discovery, the discovery must discover the NAT gateways before discovering the rest of the network, so the NAT gateways must first be found with a finder.

Network Manager is configured to trigger the seeding of all the NAT gateways if NAT translation has been enabled. However, the triggering relies on the Ping finder being active. If seeding is done, for example, using only the File finder, then the NAT gateways are not pinged even if NAT translation has been enabled. It is good practice, therefore, to seed the discovery with all the NAT gateways. You can do this using the File finder, Ping finder, or any other method.

You can also seed the discovery with NAT gateways using the Discovery Configuration GUI.

Related tasks:

"Configuring NAT translation" on page 33
To configure NAT translation to discover NAT environments, map the address-space identifier for a NAT domain to the IP address of the associated NAT gateway device.

"Enabling NAT translation" on page 120
You can set the discovery system to use NAT translation by editing $NCHOME/etc/precision/DiscoConfig.cfg to create or amend an insert into disco.NATStatus to set m_UsingNAT to 1 and m_NATStatus to 0.

Enabling NAT agents:

If you are running a NetScreen® Firewall or a Cisco® Router as a NAT gateway, you must use either the CiscoNATTelnet agent or the NATNetScreen agent.

Ensure that you enable the appropriate NAT translation agents. These agents must run to discover the NAT gateways. If they are not run, discovery cannot complete as it cannot properly discover the network without first discovering the NAT Gateways.

The NAT agents are currently CiscoNATTelnet, NATNetScreen and NATTextFileAgent. The CiscoNATTelnet agent works on Cisco IOS routers providing NAT translation and is not certified for PIX firewalls. The NATNetScreen agent is for NetScreen firewalls.

If you are using a NAT gateway other than a NetScreen Firewall or a Cisco Router, you must use the Perl agent NATTextFileAgent.pl, as described in "Enabling agents for unsupported NAT gateway devices."

Enabling agents for supported NAT gateway devices:

The CiscoNATTelnet and NATNetScreen agents connect directly to the NAT gateways to download the address mappings. You can configure these agents.

Before running these agents, you must do the following tasks:

- Enable NAT translation
- Configure trap handling

To configure and run the agents:

1. Enable the agents. There is an insert into the disco.agents table in the DiscoAgents.cfg configuration file for every installed discovery agent. To activate an agent, you must alter the insert so that the m_Valid column for that agent is set to 1. To deactivate an agent, ensure that m_Valid=0.

   The following example insert activates the CiscoNATTelnet agent.

   ```
   insert into disco.agents
   (
           m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence,
           m_DebugLevel, m_LogFile
   )
   values
   (
           'CiscoNATTelnet', 1, 8, 0, 2, 4,
           "$NCHOME/log/precision/CiscoNatTelnet.log"
   );
   ```
2. Run a discovery.

Related tasks:

"Activating agents" on page 27
You must enable the appropriate agents for the discovery you want to perform. You can specify agents for a full discovery or for a partial discovery.

Enabling agents for unsupported NAT gateway devices:

The NATTextFileAgent is provided as a backup if your NAT translation device is unsupported. You can configure this agent.

Before running the NATTextFileAgent agent, you must do the following tasks:

- Enable NAT translation
- Configure trap handling

The NATTextFileAgent reads a flat file called NATTranslations.txt that contains the NAT translations present on a particular NAT gateway. This allows the discovery an avenue to support a network containing a currently unsupported NAT gateway. This agent does not download its information from the NAT gateways, but reads a list of private to public IP address mappings from a flat file.

To configure and run the agent:

1. Install the Perl API. All Perl agents require the API to run. The API is installed by default in Network Manager.

   To check whether the API is installed, check that the following file exists:

   ```
   $NCHOME/precision/bin/ncp_perl
   ```

If the file is listed, then the Perl API is installed.

2. Create a NAT mapping file to be read by the agent that contains the public to private address mappings. Your NAT mapping file must be in a format that can be read by the agent, that is, the values must be valid IP addresses specified in columns separated by tabs.

   By default, the agent uses the file $NCHOME/etc/precision/NATTranslations.txt. If you want to create your own mappings, you must back up and edit this default file. To make the agent use the non-default NAT mapping file, edit the following line in $NCHOME/precision/disco/agents/Perlagents/NATTextFileAgent.pl:

   ```
   my $natFileName  = "$ENV{$NCHOME}/etc/precision/NATTranslations.txt";
   ```

3. The NAT mapping file contains the following columns:
   - The IP address of the NAT gateway of the NAT domain to which the device belongs. You must specify mappings for all NAT gateways in the same file.
   - The outside global address of the device, that is, the public address of the device.
   - The inside local address of the device, that is, the private address of the device.

     The following example shows a NAT mapping file for two gateways having IP addresses of 1.2.3.4 and 1.2.3.9 respectively.

     ```
     // NATGatewayIP              PublicIP                PrivateIP
     1.2.3.4                      2.3.4.5                 10.10.1.1
     1.2.3.4                      2.3.4.6                 10.10.1.2
     1.2.3.9                      2.3.6.1                 10.10.1.1
     1.2.3.9                      2.3.6.2                 10.10.1.2
     ```

   Note: From the perspective of the management station, the public IP address of a particular gateway translation is not necessarily the same as the public address that the management stations sees. The public address is the IP address that the gateway retrieves from one port and then translates and places on another port. This difference is important to note when you have chained gateways, where an IP address can be translated multiple times. The public IP is effectively the IP address that is closer to the management domain.

4. Enable the agent. There is an insert in the disco.agents table in the DiscoAgents.cfg configuration file for every installed discovery agent. To activate an agent, alter the insert so that the m_Valid column for that agent is set to 1. To deactivate an agent, ensure that m_Valid=0.

   The following example insert activates the NATTextFileAgent agent.

   ```
   insert into disco.agents
   (
          m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence, m_IsPerl
   )
   values
   (
          'NATTextFileAgent', 1, 8, 0, 2, 1
   );
   ```

5. Ensure that the NATTimer.stch stitcher has been configured to trigger a rediscovery against the NAT gateways. By default, the NATTimer.stch stitcher runs every hour. You can alter this interval by locating the following line in the stitcher file and changing the integer value:

   ```
          ActOnTimedTrigger( ( m_Interval ) values ( 1 ) ; ) ;
   ```

6. Run a discovery.

Enable agent for NAT gateway devices in private address space:

When the NAT gateway is not in the public address space, you can enable the NATGateway agent to correct the potential problem of incorrect connectivity.

The discovery assumes that the management interface of the NAT gateway is in public address space. If this is not the case, Network Manager cannot identify the address space of interfaces on the NAT gateway device, which might result in incorrect connectivity. For example, when a VPN is used to access the management interface, the NAT gateway management interface is not in the public address space.

The NATGateway agent enables Network Manager to determine whether a given interface on a NAT gateway device is on the public or private side of the NAT gateway, and thereby correctly resolve device connectivity.

To overcome this problem, activate the NATGateway agent and provide Network Manager with a mapping file, NATGateways.txt. In this file, provide a list of all NAT gateway devices together with the interfaces on each device and a field to indicate whether the interface is on the public or private side of the NAT gateway.

This agent works in conjunction with the NATGatewayRetProcessing.stch stitcher and with the NATGateways.txt file in NCHOME/precision/etc

Table 16 provides the format of the NATGateways.txt file by showing an example of the content of this file. Fields in this text file must be separated by tabs.

Table 16. Format of NATGateways.txt file

Base name	Inside or outside	Interface IP address
1.1.1.4	outside	172.16.4.10
1.1.1.4	inside	10.52.2.10
sca_T1ukP_16	outside	192.168.36.93
sca_T1ukP_16	outside	192.168.36.98

Example: Configuring a NAT discovery:

This example illustrates how to define address spaces using the NATTextFileAgent agent and how to set up associated discovery scopes.

Do the following tasks before running through the steps in this example:
- Configure the discovery to use network address translation.
- Seed the Ping finder with the IP address of each NAT gateway device.

In this example the NAT gateway devices are unsupported. This means that the NATTextFileAgent agent must be used in this NAT discovery.

The NATTextFileAgent agent uses a NAT mapping file, with the following content. There are three NAT gateway devices, with mappings for each of the devices in the associated address spaces.

```
//First NAT gateway and mappings
//NATGateway          PublicIP              Private IP
201.201.201.201       61.61.61.1             192.168.1.1
201.201.201.201       61.61.61.2             192.168.1.2
201.201.201.201       61.61.61.3             192.168.1.3
```

```
201.201.201.201         61.61.61.4             192.168.1.4
201.201.201.201         61.61.61.5             192.168.1.5
201.201.201.201         61.61.61.6             192.168.1.6

//Second NAT gateway and mappings
//NATGateway            PublicIP              Private IP
202.202.202.202         62.62.62.1             192.168.1.1
202.202.202.202         62.62.62.2             192.168.1.2
202.202.202.202         62.62.62.3             192.168.1.3
202.202.202.202         62.62.62.4             192.168.1.4
202.202.202.202         62.62.62.5             192.168.1.5
202.202.202.202         62.62.62.6             192.168.1.6

//Third NAT gateway and mappings
//NATGateway            PublicIP              Private IP
203.203.203.203         63.63.63.1             192.168.3.1
203.203.203.203         63.63.63.2             192.168.3.2
203.203.203.203         63.63.63.3             192.168.3.3
203.203.203.203         63.63.63.4             192.168.3.4
203.203.203.203         63.63.63.5             192.168.3.5
203.203.203.203         63.63.63.6             192.168.3.6
```

For the first and second address spaces private IP address space is not unique. For both of these address spaces, the private IP address space is defined by a subnet and netmask combination of 192.168.1.0/29.

Based on this NAT gateway device and address space data, define discovery scopes as follows.

1. Define each NAT gateway device and its corresponding address space. In this example, the names of the three NAT address spaces are RTP1, RTP2, and RTP3. For example, for the third NAT gateway device, the following insert defines the NAT device and its associated address space, RTP3:

   ```
   insert into translations.NATAddressSpaceIds
   (
       m_NATGatewayIP, m_AddressSpaceId
   )
   values
   (
   "203.203.203.203", "RTP3"
   );
   ```

2. Define a scope zone for each NAT gateway device.

   Note: You do not need to define a scope zone for any NAT Gateway devices whose IP address is already within any other scope zones defined for the discovery.
   For example, for the first NAT gateway device, the following insert defines the scope zone:

   ```
   insert into scope.zones
   (
       m_Protocol, m_Action, m_Zones, m_AddressSpace
   )
   values
   (
           1,
           1,
           [
               {
                   m_Subnet="201.201.201.201",
                   m_NetMask=32
   ```

```
               }
           ],
           ""
   );
   ```

3. Define scope zones for the public subnets associated with each NAT address space. For example, for the third public subnet, the following insert defines the scope zone:

   ```
   insert into scope.zones
   (
       m_Protocol, m_Action, m_Zones, m_AddressSpace
   )
   values
   (
           1,
           1,
           [
               {
                   m_Subnet="63.63.63.0",
                   m_NetMask=29
               }
           ],
           ""
   );
   ```

4. Define a scope zone for the private subnet associated with the third NAT address space only.

   Restriction: You can only define a scope zone for a private NAT address space where the subnet and netmask combination of the private subnet is unique within the discovery configuration. This excludes the first and second private subnet.
   For the third private subnet, the following insert defines the scope zone:

   ```
   insert into scope.zones
   (
       m_Protocol, m_Action, m_Zones, m_AddressSpace
   )
   values
   (
           1,
           1,
           [
               {
                   m_Subnet="192.168.3.0",
                   m_NetMask=29
               }
           ],
           "RTP3"
   );
   ```

5. Enable the NATTextFileAgent agent.

Now you can launch the NAT discovery.

Related reference:

"Quick reference for NAT discovery configuration" on page 118
Use this information as a step-by-step guide to configuring a NAT discovery..

Post-configuration NAT tasks

After you have configured NAT discoveries, you can complete several post-configuration tasks.

Tracking the progress of a NAT discovery:

During the discovery of the NAT-translation devices, you can track the discovery status in the disco.NATStatus values.

During the discovery, you initially see only the NAT-translating devices displayed in the agent despatch and returns tables. All the other data returned from the finders is stored in finders.pending database table while the discovery of the NAT-translating devices takes place.

Issue the following OQL select statement to see the discovery status:

```
select * from disco.NATStatus;
```

This statement displays a value from 0 to 4. The meaning of the value is:

- 0: NAT discovery in initial state. NAT devices have not been processed.
- 1: NAT discovery initiated. NAT gateway IPs have been sent to the Ping finder to verify their existence
- 2: NAT discovery is running.
- 3: NAT discovery processing. All the NAT gateways have been processed and the discovery is now building the `translations.NAT` table. This table ensures the correct discovery of the rest of the network.
- 4: NAT discovery complete. The entries in the finders.pending table have been moved into the finders.processing table, and the discovery continues as normal.

Use the results of this query to debug a problematic NAT discovery. The value indicates whether any discovery problems are caused by NAT or caused by the standard (non-NAT) part of the discovery process.

Debugging a NAT discovery:

To analyze a NAT discovery, use ncp_oql to follow the data through the system from the start (finders) to the end (scratchTopology) until you determine where the data is incorrect. Incorrect data indicates whether the problem is with an agent, a device, or a stitcher.

There are several queries that are useful when debugging a discovery, both NAT-based and not NAT-based.

The following OQL query indicates which agents are currently either being started (`m_State=1`), starting (`m_State=2`) or running (`m_State=3`):

```
select * from agents.status where m_State <> 0 AND m_State <> 4;
```

This query tells you which agents the current phase is waiting for in order to complete. The discovery is waiting for the agents that are meant to complete in the current phase and are in state 1, 2 or 3.

```
select * from <agentName>.despatch
where m_UniqueAddress NOT IN
      ((
      select m_UniqueAddress from <agentName>.returns where m_LastRecord = 1
      ));
```

Using the first query, you can see which agents are still running in a particular phase.

Using the following query, you can determine which entity that particular agent is processing. This can be useful in determining a problem device within your network:

```
select * from translations.ipToBaseName where m_IpAddress = '<ip>';
```

This second query shows you what base address and base name is being used for a particular IP as well as whether this IP address is considered to be in scope.

Activating the Containment Model for use with NAT:

The `NATAddressSpaceContainers.stch` stitcher creates virtual objects for each address space that contains the entities within that address space. You can activate this stitcher by uncommenting the line, // ExecuteStitcher("NATAddressSpaceContainers");, in the file `$NCHOME/precision/disco/stitchers/CreateScratchTopology.stch`.

Viewing NAT environments using Topoviz Network Views:

Using Topoviz Network Views, you can create network views based on the values of any column in the topology record of an entity. A NAT Address Spaces Dynamic Distinct view is created automatically if you activated NAT discovery as part of your discovery configuration.

As an example of viewing NAT environments, you could created a filtered network view or a Dynamic Distinct view on the following field in the NCIM topology database:

- ipEndPoint table
- addressSpace field

Note: A NAT Address Spaces Dynamic Distinct view is created automatically if **Enable Network Address Translation (NAT) Support** is turned on as part of your discovery configuration.

Chapter 3. Monitoring network discoveries

You can monitor the state and progress of your network discovery using the GUI or the command line.

Monitoring network discovery from the GUI

From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

From the Active Discovery Status page, you can also start and stop discoveries.

Related tasks:

"Starting a discovery" on page 43
After you configure a discovery, you can start and, if necessary, stop the discovery.

"Reviewing the configuration" on page 16
On the Configuration Summary window, review your settings. You can also save the settings here, and, optionally, start the discovery with the settings that you configured.

"Manually discovering a device or subnet" on page 156
You can manually discover devices so that the network topology in Network Manager matches the network.

"Starting partial discovery from the GUI" on page 158
Starting a partial discovery involves defining a seed and scopes.

Monitoring discovery progress

You can use the **Monitoring** pane to monitor the progress of the current discovery through each of the discovery phases.

To open the **Monitoring** pane, click **Discovery** > **Network Discovery Status**, and then click the **Monitoring** bar.

The following phases are shown in the table.

Interrogating Devices
During this phase, devices are first discovered by the finders, and then information is retrieved from the devices by the agents. This phase is also known as phase 1.

Resolving Addresses
During this phase, the agents resolve IP to MAC address translations. This phase is also known as phase 2.

Downloading Connections
During this phase, the switch agents download the forwarding tables from the switches in the network. This phase is also known as phase 3.

Correlating Connections
During this phase, the connectivity between the devices is calculated, the containment model is created, and the network topology is built. This phase is also known as phase -1.

You can see which phase the current discovery is in by looking at the **Status** column of the table. If a phase has not started, this column is empty. If a phase is in progress, this column shows a spinning wheel icon. If a phase has completed successfully, this column shows a green tick icon.

Status Shows the status of a particular phase. The column shows the following kinds of status.

Table 17. Discovery phase status

State	Icon	Description
Completed	✓	If a phase has completed successfully, this column shows a green tick icon.
In progress	⟳	If a phase is in progress, this column shows a spinning wheel icon.
Not started		If a phase has not started, this column is empty.

You can see how long each phase is taking in the **Elapsed Time** column in the table. Each phase takes a different amount of time depending on the scope of the discovery, the complexity of the network, and the amount of detail being retrieved from the devices. If the elapsed time continues to increase, and the work completed does not increase, the discovery might be encountering problems.

Remember: In the first phase, the count of IP addresses discovered stops increasing part way through the phase. This is part of the normal operation of the discovery. The count of IP addresses discovered only increases during the first part of the phase, while the finders discover new devices. In the latter part of the phase, the discovery agents retrieve information from these devices, and new IP addresses are not discovered.

The **Discovery Agents** section shows the progress of the discovery agents. If you think that a phase is taking too long to complete, click the **Discovery Agents** tab to see what the discovery agents are doing.

You can see the progress within a phase in the **Work Completed** column in the table. For the first phase, this column shows the number of IP addresses found so far. For the other phases, this column shows the percentage of work completed in the phase.

Related concepts:

"Discovery stages and phases" on page 280
The discovery process can be divided into two stages: data collection and data processing. The stages are subdivided into phases.

Comparing discoveries

You can use the **Monitoring** pane to compare the current discovery to the previous full discovery.

You cannot compare partial discoveries. The data in the **Previous** columns in the table is for the last full discovery.

To open the **Monitoring** pane, click **Discovery** > **Network Discovery Status**, and then click the **Monitoring** bar.

You can see the time taken to complete each phase of the previous discovery in the **Previous** subcolumn of the **Elapsed Time** column.

Note: To display discovery times from all previous discoveries, run the **`disco_profiling_data.pl`** script from the command line. For more information on the **`disco_profiling_data.pl`** script, see the *IBM Tivoli Network Manager IP Edition Administration Guide*.

The time taken for each phase depends on the scope of the discovery, the complexity of the network, and the amount of detail being retrieved from the devices. If the network has not changed significantly, and the discovery scope and settings have not changed significantly, but the elapsed time for a phase in the current discovery is significantly more than the time taken for the same phase in the previous discovery, the discovery might be encountering problems.

You can see how many IP addresses are being found in the current discovery and how many were found in the previous discovery in the **Work Completed** column in the table. If significantly fewer IP addresses have been found in the current discovery, there could be a problem with the scope of the discovery, or with SNMP access to devices.

Monitoring ping finder progress

You can use the **Ping Finder Status** table to monitor the progress of the ping finder during a discovery.

To open **Ping Finder Status**, click **Discovery** > **Network Discovery Status**, and then click the **Ping Finder Status** tab.

You can use the **Ping Finder Status** table to see which IP addresses and subnets have been discovered up to this point. If the ping finder is currently processing a subnet, you can also see which IP address was last pinged.

The **Ping Finder Status** table contains the following information:

Address
A list of IPs and subnets discovered to this point.

Netmask
For each subnet, this column indicates the netmask value.

Last Pinged
The last IP address pinged.

Status
Indicates whether the Ping finder is still pinging this device or subnet or whether it has completed pinging.

Table 18. Ping finder status

State	Icon	Description
Completed	✅	Ping finder has completed the pinging of this subnet or IP address.
Started	➡	Ping finder is currently pinging this subnet or IP address.
Stopped	🟩	Ping finder has not started pinging this subnet or IP address.
Awaiting status	⟳	System is awaiting Ping finder status for this subnet or IP address.

Monitoring discovery agent progress

You can use the **Agents Status** section to monitor the progress of the discovery agents through each of the discovery phases.

Discovery agents gather data from discovered devices. This data is used during the Correlating connectivity phase of the discovery (phase -1) to build the network connectivity and containment.

You can use the **Agents Status** to answer these and other questions as the discovery is running:

- Are all agents running okay?
- Have any agents failed?
- Are any agents failing to complete?
- Which device is a particular agent currently working on?

1. To open **Agents Status**, click **Discovery** > **Network Discovery Status**, then click the **Agents Status** tab. The **Agents Status** section contains two tables, the **Agents Status** table at the top and the **IP Address Status** table below. The **Agents Status** table toolbar contains the following controls.

   Filter Agents by Phase
   Use the phase drop-down list to select a discovery phase. The agents table then displays all discovery agents that have started during the current discovery and that are scheduled to finish in the discovery phase that you selected.

   Refresh 🔄
   Refreshes the data in both the Agents Status and IP Address Status table. The icon changes to the **Refreshing** icon ✳ while the table data is being refreshed. You cannot refresh the tables again until the refresh has completed.

   The **Agents Status** table lists all the agents that have started so far during this discovery and contains the following information. This information is updated every 20 seconds. When you first open this table, it is sorted by descending order of **State**.

   Agent Discovery agents that have started during the current discovery and that are scheduled to finish in the discovery phase that you selected.

   Completes in Phase
   The phase in which the discovery agent completes.

   State Current state of the discovery agent. Possible states, in the default descending order, are listed in the following table.

Table 19. Agent states

State	Value	Icon	Description
Died	5	✖	The agent has terminated unexpectedly. This is a potential discovery problem.
Finished	4	✓	The agent is still running but has finished processing of all the IP addresses in its queue. The agent is still available to process any further agents placed in the queue.
Running	3	✳	The agent is currently processing IP addresses.

Table 19. Agent states (continued)

State	Value	Icon	Description
Starting	2	✳	The agent is starting up.
Not running	1	▣	The Agent is not running.

Total IP Addresses

The total number of IP addresses that this agent is required to process. This number increases as the discovery progresses and the finders discover more devices that need to be processed by the agent.

Outstanding IP Addresses

The number of IP addresses that are waiting to be processed by this agent. This number can go up and down during the discovery. The number increases initially as the discovery progresses and the finders discover more devices that need to be processed by the agent. As the agent completes processing on IP addresses, this number decreases until it reaches zero.

Note: If this value does not go down to zero during the discovery, this means that the agent was unable to complete processing on one or more IP addresses and there is a potential discovery problem.

2. Click an agent in the **Agents Status** table. The **IP Address Status** table lists the IP addresses that have been processed or are currently being processed by this agent. The **IP Address Status**table responds to changes in the **Agents Status** table. The table updates in the following situations: when a new agent is selected in the **Agents Status** table; when changing the filtering of the **IP Address Status** table by **All** or **Queue**; and when the **Agents Status** table **Refresh** 🔄 button is pressed. When you first open this table, it is sorted by descending order of **State**.

   Agent_name

   Use this radio button to specify whether to display all IP addresses (All) or only IP addresses queued for processing (Queue). The default setting is Queue.

   All Set the **Details** table to display all IP addresses for this agent. This includes IP addresses that have been queued for processing by the agent, IP addresses currently being processed by the agent, and IP addresses that have already been processed by the agent.

   Queue

   Set the **Details** table to display only IP addresses that have been queued for processing by this agent.

   IP Address

   IP addresses processed by this agent. If **All** is selected, then this column displays IP addresses processed, in processing, or queued for processing by this agent. If **Queued** is selected, then this column displays IP addresses queued for processing by this agent.

   State Current state of the IP address. Possible states, in the default descending order, are listed in the following table:

Table 20. IP address states

State	Value	Icon	Description
Died	5	⊗	Processing of the IP address terminated unexpectedly. This is a potential discovery problem.
Finished	4	✓	An agent has completed processing this IP address.
Running	3	⟳	An agent is currently processing this IP address.
Starting	2	⟳	An agent is beginning to process this IP address.
Not running	1	■	This IP address is not currently being processed.

Elapsed Time
The time taken for the agent to process this IP address, expressed in the format HH:MM:SS. This value is only displayed for those IP addresses that have completed processing.

Despatch Time
The date and time at which the agent began processing this IP address. This value is only displayed for those IP addresses for which processing has begun or completed.

Return Time
The date and time at which the agent retrieved data for this IP address. This value is only displayed for those IP addresses that have completed processing.

SNMP Access
Indicates whether the agent was able to access this IP address using SNMP.

Related tasks:

"Troubleshooting an unusually long discovery" on page 165
A discovery might be taking a long time to complete because an agent is unable to complete processing on a specific device. Use the **Agents Status** section to determine which agent is taking a long time to complete and which device it is working on.

"Identifying failed agents" on page 167
A source of discovery failure can be agents that terminate unexpectedly during discovery. Use the **Agents Status** section to determine if any agents have terminated unexpectedly.

Monitoring discovery from the command line.

When the **ncp_disco** process is running, you can monitor the progress of the discovery by using the OQL Service Provider, the **ncp_oql** process, to query the discovery databases to determine what is happening at any time.

The queries demonstrated in the subsequent topics have been generalized for all discovery scenarios and are not limited to the layer 3 discovery.

The examples are given only to demonstrate the amount of flexibility you have when retrieving information from databases using OQL. Using the schematic definitions of all the databases and knowledge of OQL syntax, you can construct queries that answer any questions you have regarding the current status of the discovery process.

You can issue simple queries to find out, for example, what the **ncp_disco** process is currently doing, which discovery agents have discovered devices, or how many devices have been discovered so far. You can also issue complex queries to find out, for example, which devices have been discovered by a specific discovery agent, or which discovery agents have interrogated a specific device.

For information on starting the OQL Service Provider, including prerequisites, see the *IBM Tivoli Network Manager IP Edition Language Reference*.

Related tasks:

"Discovering the network using the command-line interface" on page 47
As an experienced user, you can configure and track a network discovery using configuration files and database queries.

Sample discovery status queries

You can use queries similar to these examples to find out the status of different parts of the discovery.

Sample: Determining which address the Ping finder is pinging

The following query returns the current address being pinged by the Ping finder:

```
select m_CurrentAddress from pingFinder.status;
go
.
{
                m_CurrentAddress=192.168.0.1;
}
```

Sample: Identifying the current phase of the discovery

The following example shows how to identify the current phase of the discovery. The results of the above query show that the discovery process is still in data collection phase 1.

```
select * from disco.status;
go
.
{
                m_DiscoveryMode=0;
                m_Phase=1;
                m_BlackoutState=0;
                m_CycleCount=0;
                m_ProcessingNeeded=0;
                m_FullDiscovery=0;
}
```

Sample: Identifying the status of a NAT discovery

This example shows how to identify the status of the NAT discovery.

```
select m_NATStatus from disco.NATStatus;
go
.
{
                m_NATStatus=3;
}
```

Sample: Identifying which agents are enabled

This example shows how to identify whether you have enabled the appropriate discovery agents.

```
select m_AgentName, m_Valid from disco.agents
where m_Valid = 1;
go
...
{
                m_AgentName='Details';
                m_Valid=1;
}
{
                m_AgentName='AssocAddress';
                m_Valid=1;
}
{
                m_AgentName='IpRoutingTable';
                m_Valid=1;
}
{
                m_AgentName='IpForwardingTable';
                m_Valid=1;
}
```

Sample: identifying the status of the discovery stitchers

The following example shows how to identify the status of the stitchers by querying the stitchers.status table.

```
select * from stitchers.status
where m_State > 0 ;
go
.........
{
                m_Name='AgentRetToInstrumentationSubnet';
                m_State=3;
}
{
                m_Name='DetailsRetProcessing';
                m_State=3;
}
.....
.....
{
                m_Name='DetectionFilter';
                m_State=3;
}
{
                m_Name='FnderProcToDetailsDesp';
                m_State=3;
}
{
                m_Name='FnderRetProcessing';
                m_State=3;
}
```

The results from the query show the current status of all stitchers that have been called by the discovery process so far. Note that the results shown above have been abbreviated.

Sample: identifying which agents are active

The following example shows how to query the status of the agents in the agents database.

```
select * from agents.status
where m_State > 0 ;
go
..
```

```
{
                m_Name='Details';
                m_State=3;
                m_NumConnects=1;
}
{
                m_Name='IpRoutingTable';
                m_State=3;
                m_NumConnects=1;
}
```

The results of the above query show that only the Details agent and the IpRoutingTable agent are active (that is, they have a state greater than zero).

Related reference:

Appendix A, "Discovery databases," on page 183
There are various specialized databases that are used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.

Sample device queries

You can use queries similar to these examples to identify devices that meet certain criteria, for example, devices that have been found by the finders.

Sample: Identifying which devices have been found by the finders

The following example shows how to identify devices that have been found by the finders.

```
select * from finders.returns;
go
....
{
                m_UniqueAddress='172.20.12.253';
                m_Protocol=1;
                m_Creator='IpRoutingTable';
}
{
                m_UniqueAddress='172.20.22.61';
                m_Protocol=1;
                m_Creator='IpRoutingTable';
}
{
                m_UniqueAddress='172.20.0.221';
                m_Protocol=1;
                m_Creator='IpRoutingTable';
}
{
                m_UniqueAddress='10.10.35.17';
                m_Creator='PingFinder';
}
```

The above query shows the devices discovered by the Ping finder as well as devices reported as a result of connections discovered by the `IpRoutingTable` Discovery agent.

Sample: Identifying devices that have been sent to the Details agent

The following example shows how to identify devices that have been sent to the Details agent.

```
select * from Details.despatch;
go
.................................................................
................................
{
                m_UniqueAddress='10.10.38.82';
}
{
                m_UniqueAddress='10.10.38.83';
}
.....
.....
{
                m_UniqueAddress='10.10.38.84';
}
{
                m_UniqueAddress='10.10.38.87';
}
{
                m_UniqueAddress='10.10.38.88';
}
{
                m_UniqueAddress='10.10.38.89';
}
{
                m_UniqueAddress='10.10.38.90';
}
```

Sample: Identifying devices that have been returned from the Details agent

To identify which devices have returned from the Details agent, query the `returns` table of the Details agent, as shown below.

```
select * from Details.returns;
go
.................................................................
................................
{
                m_UniqueAddress='10.10.8.255';
                m_UpdAgent='Details';
                m_HaveAccess=1;
                m_Description='Ascend Max-HP T1/PRI S/N;
                m_ObjectId='1.3.6.1.4.1.529.1.2.6';
                m_LastRecord=1;
}
{
                m_UniqueAddress='10.10.9.1';
                m_UpdAgent='Details';
                m_Name='minotaur.Kazeem.San.COM';
                m_HaveAccess=0;
                m_LastRecord=1;
}
.....
.....
{
                m_UniqueAddress='10.10.9.2';
                m_UpdAgent='Details';
                m_Name='cyclops.Kazeem.San.COM';
                m_HaveAccess=0;
                m_LastRecord=1;
}
{
                m_UniqueAddress='10.10.9.3';
                m_UpdAgent='Details';
                m_Name='centaur.Kazeem.San.COM';
                m_HaveAccess=0;
                m_LastRecord=1;
}
```

Sample: Identifying all devices discovered until now

The following example shows how to identify all known network entities.

```
select m_Name, m_ObjectId, m_UniqueAddress
from workingEntities.finalEntity;
go
.................................
{
                m_Name='10.10.8.255';
                m_ObjectId='1.3.6.1.4.1.529.1.2.6';
                m_UniqueAddress='10.10.8.255';
}
{
                m_Name='minotaur.Kazeem.San.COM';
                m_UniqueAddress='10.10.9.1';
}
.....
.....
{
                m_Name='cyclops.Kazeem.San.COM';
                m_UniqueAddress='10.10.9.2';
}
```

Sample: Identifying which agents have discovered devices

The following example shows how to identify the agents that have discovered devices.

```
select m_Name, m_Creator
from workingEntities.finalEntity;
go
.................................
{
                m_Name='b11-m1-2611.Kazeem.San.COM[ 0 [ 2 ] ]';
                m_Creator='IpRoutingTable';
}
{
                m_Name='b-ayo.Kazeem.San.COM';
                m_Creator='Details';
}
{
                m_Name='b11-m1-2611.Kazeem.San.COM[ 0 [ 1 ] ]';
                m_Creator='IpRoutingTable';
}
.....
.....
{
                m_Name='b11-m1-2611.Kazeem.San.COM';
```

Sample network entity queries

You can use queries on the instrumentation database to identify whether network entities such as subnets and VLANs have been discovered. The instrumentation database tables store a record of every discovered device.

Sample: Identifying the number of discovered subnets

The following example query returns details of the discovered subnets.

```
select * from instrumentation.subNet;
go
.......................................
{
                m_SubNet='172.20.67.0';
                m_NetMask='255.255.255.0';
}
.....
.....
{
                m_SubNet='172.20.70.0';
                m_NetMask='255.255.254.0';
}
{
                m_SubNet='172.20.95.0';
                m_NetMask='255.255.255.0';
}
( 81 record(s) : Transaction complete )
```

Sample: Identifying discovered VLANs

The following example query returns details of the discovered VLAN IDs.

```
select * from instrumentation.vlan;
go
.......................................
{
                m_Vlan=23;
}
{
                m_Vlan=65;
}
.....
.....
{
                m_Vlan=677;
}

( 4826 record(s) : Transaction complete )
```

Sample complex discovery queries

You can use queries similar to these examples to identify devices that meet certain criteria, for example, devices that have been found by particular discovery agents.

Identifying which devices have been discovered by a particular agent

The following example query identifies which devices have been discovered by the IpRoutingTable agent.

```
select m_Name, m_Creator
from workingEntities.finalEntity
where
m_Creator = 'IpRoutingTable';
```

```
go
.................................
{
                m_Name='10.10.63.194';
                m_Creator='IpRoutingTable';
}
.....
.....
{
                m_Name='b11-m1-2611.Kazeem.San.COM[ 0 [ 1 ] ]';
                m_Creator='IpRoutingTable';
}
{
                m_Name='b11-m1-2611.Kazeem.San.COM';
                m_Creator='IpRoutingTable';
}
```

Identifying devices that have been sent to a specific agent

The following example query identifies devices that have been sent to the IpRoutingTable agent.

```
select m_Name, m_ObjectId, m_Description
from IpRoutingTable.despatch;
go
.................................
{
       m_Name='10.10.63.193';
       m_ObjectId='1.3.6.1.4.1.9.1.108';
       m_Description='Cisco Internetwork Operating System Software
IOS (tm) 7200 Software (C7200-JS-M), Version 12.0(4)T,  RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1999 by Cisco Systems, Inc.
Compiled Thu 29-Apr-99 06:27 by kpma';
}
.....
.....
{
       m_Name='10.10.71.248';
       m_ObjectId='1.3.6.1.4.1.9.1.258';
       m_Description='Cisco Internetwork Operating System Software
IOS (tm) MSFC Software (C6MSFC-IS-M), Version 12.0(7)XE1, EARLY DEPLOYMENT
RELEASE SOFTWARE (fc1)
TAC:Home:SW:IOS:Specials b-ayo k-az-eem for info
Copyright (c) 1986-2000 by Cisco Systems, Inc.
Compiled Fri 04-Feb-00 00:';
}
```

Identifying devices that have been returned by a specific agent

The following example query identifies devices returned by the IpRoutingTable discovery agent.

```
select m_Name from IpRoutingTable.returns;
go
.................................
{
                m_Name='10.10.71.248';
}
.....
.....
{
                m_Name='10.10.71.248';
}
{
                m_Name='10.10.71.248';
}
```

Identifying additional devices that have been discovered by a specific agent

An agent can discover additional devices by interrogating a device. In this situation, the additional device would be in the returns table of that agent, but not the despatch table. You can identify which devices are present in the IpRoutingTable.returns table, but not in the IpRoutingTable.despatch table, by performing a join between the IpRoutingTable.despatch and IpRoutingTable.returns tables, as in the following example.

```
select IpRoutingTable.returns.m_Name from
IpRoutingTable.returns, IpRoutingTable.despatch
where
IpRoutingTable.returns.m_Name <>
IpRoutingTable.despatch.m_Name;
go
.........................................
{
                m_Name='10.10.71.237';
}
.....
.....
{
                m_Name='10.10.71.55';
}
{
                m_Name='10.10.71.51';
}
```

Identifying the devices that an agent has enqueued

The following example returns those devices in the despatch table that have not yet been returned.

```
select * from <agent>.despatch
where
(
 m_UniqueAddress NOT IN
  (( select m_UniqueAddress from <agent>.returns where m_LastRecord = 1 ))
);
```

Sample queries for locating a specific device

To see whether a specific device has been discovered, you can use queries similar to these examples to search through the discovery data flow.

Sample: Identifying whether a device is present in the workingEntities database

The following example query determines if the device is present in the workingEntities database.

```
select * from workingEntities.finalEntity
where m_UniqueAddress ='10.10.63.239';
go
.
( 0 record(s) : Transaction complete )
```

Sample: Identifying whether a device has been returned from the AssocAddress agent

If the device is not present in the workingEntities database, you can use the following example query to determine if the device has been returned from the AssocAddress agent.

```
select * from AssocAddress.returns
where m_UniqueAddress = '10.10.63.239';
go
.
( 0 record(s) : Transaction complete )
```

Sample: Identifying whether a device has been returned from the Details agent

If the device has not been returned from the AssocAddress agent, you can use the following example query to determine if the device has been returned from the Details agent.

```
select * from Details.returns
where m_UniqueAddress = '10.10.63.239';
go
.
( 0 record(s) : Transaction complete )
```

Sample: Identify whether a device has been sent to the Details agent

If the device has not been returned from the Details agent, you can check if the device has been sent to the Details agent by querying the `Details.despatch` table, as shown below. This result indicates that the device has been sent to the Details agent, but has not yet been processed.

```
select * from Details.despatch
where m_UniqueAddress='10.10.63.239';
go
.
{
                m_UniqueAddress='10.10.63.239';
}
( 1 record(s) : Transaction complete )
```

Sample: Identifying whether a device has been discovered by the finders

If the device is not in the `Details.despatch` table, you can query the `finders` database, as shown below. This result shows that the device has been discovered by the finders.

```
select * from finders.processing
where m_UniqueAddress='10.10.63.239';
go
.
{
                m_UniqueAddress='10.10.63.239';
}
( 1 record(s) : Transaction complete )

select * from finders.returns
where m_UniqueAddress='10.10.63.239';
go
.
( 0 record(s) : Transaction complete )
```

Chapter 4. Classifying network devices

On completion of discovery, Network Manager IP Edition automatically classifies all discovered network devices based on a predefined device class hierarchy. You can change the way network devices are classified.

Changing the device class hierarchy

Change the device class hierarchy to change the way network devices are classified. A common situation that requires a change to the class hierarchy is when the discovery process identifies an unclassified device, that is, a device that is not defined in the class hierarchy.

Following a discovery, you can check whether any devices are unclassified by running the following reports:

- Devices with Unclassified SNMP Object IDs report
- Devices with Unknown SNMP Object IDs report

Listing the existing device classes

Before you edit AOC definitions and reinstantiate the topology, list the device classes that are currently in use.

You list existing device classes by querying the ncp_model databases. The query returns the names of the AOCs to which devices in the current topology have been instantiated. Substitute your domain name and username where NCOMS and admin are specified.

1. Log into the OQL Service Provider using the following command:

   `ncp_oql -domain` *`NCOMS`* `-username` *`admin`* `service` *`Model`* You can also issue this query using the Management Database Access page.
2. Specify the relevant password when prompted.
3. Type the following query:

   `select ClassName from master.entityByName;`

   `go` Here is an example of the output of this query:

   ```
   {
    ClassName='Device';
   }
   {
    ClassName='Interface';
   }
   .....
   .....
    ClassName='MainNode';
   }
   {
    ClassName='CiscoSwitch';
   }
   ( 131 record(s) : Transaction complete )
   ```

Creating and editing AOC files

Create and edit AOC files to classify unclassified devices or to change the class hierarchy of your topology.

If the discovery process identified an unclassified device, you can classify the device by creating a new AOC file that is specific to the device class to which this device belongs.

You can edit AOCs in either of two ways: update the ncp_class databases, or modify the AOC file definitions:

- If you want to update the ncp_class database (and therefore the current AOC definitions) directly, then use the Management Database Access or the OQL Service Provider.
- If you want to modify the AOC file definitions, then follow the steps in this topic.

1. Go to the `NCHOME/precision/aoc` directory.
2. Back up any files that you want to edit.
3. Stop all running Network Manager processes.
4. Create a new text file or edit an existing AOC file using a text editor.

   Restriction: Only alphanumeric characters and the underscore (_) character may be used for AOC filenames. Any other characters, for example the hyphen (-) are forbidden.
5. If you created a new AOC file, add a new insert to the class.classIds database table in the `ClassSchema.cfg` configuration file.
6. Edit the startup options for the **`ncp_class`** process and set the -read_aocs_from option to ensure that the new or changed AOC files are read.
7. Start all Network Manager processes.
8. Ensure that a domain-specific version of any new AOC files is present in the `NCHOME/precision/aoc` directory.
9. Back up and remove the discovery cache files in the `NCHOME/var/precision` directory.
10. Run a full discovery and check that the results match the changes you made.

Related reference:

"AOC specific to device class" on page 152
Use this sample AOC file to understand how Network Manager assigns discovered devices to the device class at a lower level in the class hierarchy.

Applying AOC changes to the topology and to the reports

After you have updated the AOC definitions and passed the changes to **`ncp_class`**, you can apply the changes to the topology by waiting for the next discovery to complete or by restarting the discovery at the point when the topology is passed from ncp_disco to ncp_model.

When the next full discovery completes, the AOC changes that you made are automatically applied to the network topology.

If you do not want to wait for the next full discovery, use the appropriate stitcher to restart the discovery at the required point. To re-instantiate the containment model, you must start the stitcher that sends the scratch topology from **`ncp_disco`** to **`ncp_model`**.

1. Log into the OQL Service Provider or access the Management Database Access.
2. Issue the following query to the disco.status table to confirm that the **ncp_disco** process is in rediscovery mode: `select * from disco.status;`

   Here is a sample response.

   ```
   m_DiscoveryMode=1;
   m_Phase=1;
   m_BlackoutState=0;
   m_CycleCount=0;
   m_ProcessingNeeded=0;
   m_FullDiscovery=0;
   ```

   From the results returned by the query, you can see that **ncp_disco** is currently in the rediscovery mode, that is, m_DiscoveryMode=1.
3. Start the SendTopologyToModel stitcher. The SendTopologyToModel sends the scratch topology from ncp_disco to ncp_model.
   a. Ensure that you are in the OQL Service Provider or the Management Database Access.
   b. To insert the stitcher into the stitchers.actions table, issue the following command:

      ```
      insert into stitchers.actions
      ( m_Name )
      values
      ( 'SendTopologyToModel' );
      ```

   After your OQL insert is accepted, the stitcher is invoked and the network topology is sent to ncp_model. When the topology is sent, it is instantiated in accordance with the modified AOC hierarchy.
4. In order to ensure that the newly classified devices are removed from the Devices with Unclassified SNMP Object IDs report and the Devices with Unknown SNMP Object IDs report, perform the following steps:
   a. Clarify exactly which new sysObjectId values are being mapped by the new or edited AOC files. For example, the original AOC files mapped the following sysObjectId values:
      - 1.2.3.4
      - 1.5.6.*

      Then two new sysObjectId values are added to the system: 1.9.8 and 1.5.6.7. In the AOC file, the sysObjectId value 1.5.6.7 is covered by the mapping 1.5.6.*. However, the AOC file must be updated to add the sysObjectId value 1.9.8.
   b. Clarify which AOC files are mapped by the NCIM topology database mappings table. The mappings table is used by the Devices with Unclassified SNMP Object IDs report and the Devices with Unknown SNMP Object IDs report to determine which data to show in the reports. This table is not automatically updated when you edit AOC files and restart the Topology manager, `ncp_class`, and consequently these reports continue to show the new sysObjectId values as unclassified and unknown. The mappings in the mappings table are also more specific than the mappings in the AOC files. For example, the NCIM topology database mappings table might contain the following data:

Table 21. Example of data from the NCIM topology database mappings table

mappingGroup	mappingKey	mappingValue	Description
sysObjectId	1.2.3.4	Device Type A	Description of Device Type A

Table 21. Example of data from the NCIM topology database mappings table (continued)

mappingGroup	mappingKey	mappingValue	Description
sysObjectId	1.5.6.1	Device Type B	Description of Device Type B
sysObjectId	1.5.6.2	Device Type C	Description of Device Type C

In the AOC file, only the sysObjectId value 1.9.8 needed to be added because the generic mapping 1.5.6.* covered the new sysObjectId value 1.5.6.7. However, in the NCIM topology database mappings table both sysObjectId values 1.9.8 and 1.5.6.7 must be added.

c. From the command line, update the NCIM topology database mappings table with relevant records for the new sysObjectId values. For example, to add records for the two new sysObjectId values 1.9.8 and 1.5.6.7, issue the following SQL insert statements:

```
insert into mappings (mappingGroup, mappingKey, mappingValue)
values ('sysObjectId', '1.9.8', 'device_type');
insert into mappings (mappingGroup, mappingKey, mappingValue)
values ('sysObjectId', '1.5.6.7', 'device_type');
```

Where *device_type* is the device type to which the sysObjectId value must be mapped. For information on the NCIM topology database mappings table, see the *IBM Tivoli Network Manager IP Edition Topology Database Reference*.

After the AOC changes have been applied to the topology, either automatically by waiting for the next discovery, or manually by performing the steps in this topic, you will notice the following changes are applied to network polling and visualization.

- When you define a new polling policy, the new classes you defined are displayed in the Classes tab in the Poll Policy Editor.
- When you visualize the network using network views, the network view tree now displays the classes defined in the modified class hierarchy.
- If you updated the NCIM topology database mappings table as described then the Devices with Unclassified SNMP Object IDs report and the Devices with Unknown SNMP Object IDs report no longer return any devices.

AOC file samples

Use the AOC file samples to understand how Network Manager assigns discovered devices to the device classes in the class hierarchy.

EndNode class

Use this sample EndNode class AOC file to understand how Network Manager assigns discovered devices to the EndNode class.

Sample

The following sample AOC file fragment assigns devices to the EndNode class using the filter defined in the instantiate_rule clause.

```
//***********************************************************
//
// File : EndNode.aoc
//
//***********************************************************
active object 'EndNode'
{
```

```
super_class = 'Core';
instantiate_rule = "EntityOID like '1 \.3\.6\.1\.4\.1\.2021\.' OR
EntityOID = '1.3.6.1.4.1.2021' OR
EntityOID = '1.3.6.1.4.1.1575' OR
EntityOID like '1 \.3\.6\.1\.4\.1\.11\.2\.3\.9\.' OR
EntityOID = '1.3.6.1.4.1.11.2.3.9' OR
(EntityType = 1 AND EntityOID IS NULL)
OR
...
OR
(
EntityOID = '1.3.6.1.4.1.1977'
)
OR
(
EntityOID like '1\.3\.6\.1\.4\.1\.2136\.'
)
OR
...
```

For the EndNode class the instantiate_rule is very long. It consists of multiple lines comparing the EntityOID, (this is the sysObjectID of the device), to various values, joined together by an OR operator. There are different versions of the OR comparison:

EntityOID = '1.3.6.1.4.1.2021'

This filter is looking for an exact match of the EntityOID to the value 1.3.6.1.4.1.2021. If the match is not exact, then the comparison fails and the device is not assigned to the EndNode class.

EntityOID like '1\.3\.6\.1\.4\.1\.11\.2\.3\.9\.'

This filter is looking for a match like the value 1\.3\.6\.1\.4\.1\.11\.2\.3\.9\. The \. is required to make sure that the . (period) is matched. Also, notice that the value ends in \. This allows matching OIDs that start with the specified value but have additional values following the last . (period) specified.

NetworkDevice class

Use this sample NetworkDevice class AOC file to understand how Network Manager assigns discovered devices to the NetworkDevice class.

Sample

The following sample AOC file fragment assigns devices to the NetworkDevice class using the filter defined in the instantiate_rule clause.

```
//************************************************************
//
// File : NetworkDevice.aoc
//
//************************************************************
active object 'NetworkDevice'
{
super_class = 'Core';
instantiate_rule = 'EntityType = 1 OR // Chassis
EntityType = 2 OR // Interface
EntityType = 3 OR // LogicalInterface
EntityType = 5 OR // Card
EntityType = 6 OR // PSU
EntityType = 8 OR // Module
EntityType = 0';
...
```

For the NetworkDevice class, the instantiate_rule tries to match device types. The following examples are filters that are used in the instantiate_rule.

EntityType = 1
Matches all entities discovered that are chassis devices. Chassis devices have the field entityType set to a value of 1 in the NCIM topology database entityData table.

EntityType = 2
Matches all entities discovered that are ports or interfaces. Ports and interfaces have the field entityType set to a value of 2 in the NCIM topology database entityData table.

EntityType = 3
Matches all entities discovered that are logical interfaces. Logical interfaces have the field entityType set to a value of 3 in the NCIM topology database entityData table.

EntityType = 5
Matches all entities discovered that are cards. Cards have the field entityType set to a value of 5 in the NCIM topology database entityData table.

EntityType = 6
Matches all entities discovered that are power supply units (PSUs). PSUs have the field entityType set to a value of 6 in the NCIM topology database entityData table.

EntityType = 8
Matches all entities discovered that are modules. Modules have the field entityType set to a value of 8 in the NCIM topology database entityData table.

AOC specific to device class

Use this sample AOC file to understand how Network Manager assigns discovered devices to the device class at a lower level in the class hierarchy.

Sample

The following sample AOC file fragment assigns devices to the EWindowsNetHarmoni class using the filter defined in the instantiate_rule clause. This is an EndNode device.

```
//*************************************************************
//
// File : EWindowsNetHarmoni.aoc
//
//*************************************************************
active object 'EWindowsNetHarmoni'
{
super_class ='EndNode';

instantiate_rule = "EntityOID like '1 \.3\.6\.1\.4\.1\.1977\.1\.6\.1279\.'";
...
```

For the EWindowsNetHarmoni class, the following parameters are defined in the AOC file. The instantiate_rule parameter is long. It consists of multiple lines comparing the EntityOID, (this is the sysObjectID of the device), to various values, joined together by an OR operator. There are different versions of the OR comparison:

super_class ='EndNode'

This parameter establishes the device as belonging to the EndNode class. The EWindowsNetHarmoni class inherits all the attributes of the EndNode class.

instantiate_rule = "EntityOID like '1 \.3\.6\.1\.4\.1\.1977\.1\.6\.1279\.'"

This filter is looking for a match to the value 1\.3\.6\.1\.4\.1\.11\.2\.3\ .9\. The \. is required to make sure that the . (period) is matched. Also, notice that the value ends in \. This allows matching OIDs that start with the specified value but have additional values following the last . (period) specified.

Chapter 5. Keeping discovered topology up-to-date

After a discovery has completed, you can keep the discovered topology updated by scheduling a discovery, configuring automatic discovery, manually discovering devices, and removing devices.

Scheduling a discovery

After a full discovery has completed, you can schedule further discoveries by editing the `FullDiscovery.stch` file.

To schedule a discovery using the `FullDiscovery.stch` file:

1. Back up and edit the `NCHOME/disco/stitchers/FullDiscovery.stch` file.
2. Uncomment one of the ActOnTimedTrigger lines, and modify it to run the discovery at a certain time. As an example, to schedule the discovery for 11:00 PM every day, modify the line as follows:

   ```
   ActOnTimedTrigger(( m_TimeOfDay ) values ( 2300 ) ; );
   ```
3. Save and exit the file.

Here are several other examples of scheduling a discovery.

- To schedule a discovery on the sixth day of the week since Sunday (Saturday) at 11 PM (where Sunday = 0, Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5, Saturday = 6):

  ```
  ActOnTimedTrigger(( m_DayOfWeek , m_TimeOfDay )
   values ( 6 , 2300 ) ; ) ;
  ```
- To schedule a discovery on the thirteenth day of each month at 2 PM:

  ```
  ActOnTimedTrigger(( m_DayOfMonth , m_TimeOfDay )
  values ( 13 , 1400 ) ; );
  ```
- To schedule a discovery at intervals of 13 hours:

  ```
  ActOnTimedTrigger(( m_Interval ) values ( 13 ) ; );
  ```

Related concepts:

"About types of discovery" on page 1
Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Configuring automatic discovery

Network Manager provides a mechanism to automatically trigger a partial discovery based on receipt of a trap. This is performed by the Disco plug-in to the Event Gateway. Device traps might indicate a change in a network device or the presence of a new network device.For more information on the Disco plug-in, see the *IBM Tivoli Network Manager IP Edition Event Management Guide*.

Related concepts:

"About types of discovery" on page 1
Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Manually discovering a device or subnet

You can manually discover devices so that the network topology in Network Manager matches the network.

Sometimes you might know that one or more devices have had their configuration changed, and want to discover them again regardless of whether the system has detected the change from traps sent by the devices.

You can manually discover a device or subnet in the following ways:

- You can use the Discovery Configuration GUI to specify individual devices or complete subnets to be discovered.
- You can discover specific devices or sets of devices from the Hop View or Network Views.
- You can make inserts to the finders.rediscovery table using ncp_oql, specifying the IP address or subnet to be discovered.

Note: Do not use manual discoveries to remove devices from the topology. Devices that are no longer accessible remain in the topology until their LingerTime reaches zero and another discovery is run. Do manual discoveries only against devices that are operational but whose configuration has been changed.

Related tasks:

"Removing a device from the network" on page 160
You can manually remove a device that is known to be scheduled for permanent removal from the network.

"Monitoring network discovery from the GUI" on page 131
From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

Manually discovering a device or subnet using the GUI

You can configure and launch discovery of a device or subnet from the Discovery Configuration GUI. You can customize the discovery configuration to make partial discovery run as quickly as possible.

Enabling partial discovery agents

You can configure partial discovery by enabling the appropriate agents from the **Partial Discovery Agents** tab in the Discovery Configuration GUI.

You can speed up the time taken for a partial discovery by selecting only those agents essential to discover new or modified devices.

Related reference:

"Guidance for selecting agents" on page 336
To discover device technologies (that is, those that use protocols other than IP) on your network, you must ensure that the appropriate agents are active.

Configuring advanced partial discovery settings

Among the advanced partial discovery settings that you can configure are feedback, rebuilding of layer, and remote neighbor parameters.

Configuring feedback settings:

You can specify feedback settings when configuring a partial discovery with the GUI.

Feedback is the mechanism by which data returned by agents is used to find other devices. Examples of feedback data include the IP address of remote neighbors, or the subnet within which a local neighbor exists.

The feedback mechanism allows any new IP addresses to be fed back into the discovery and thus increases the size of the discovered network. You need to balance completeness of the discovered topology (feedback *on*) with greater speed of discovery (feedback *off*).

You can choose from the following options after selecting the **Advanced** tab within the Configuration option in the Discovery Configuration GUI:

- **No Feedback**: Feedback is off for all discoveries. This option provides speed but discovers only those devices specified to the finders, and hence provides an incomplete topology. However, this setting ensures that discoveries complete in the quickest possible time.
- **Feedback**: Feedback is on for full discoveries and partial discoveries. This option provides a complete topology in all situations but takes the longest time.
- **Feedback Only on Full**: Feedback is on for full discoveries, ensuring a complete topology. For partial discoveries there is no feedback. This ensures that the partial discovery runs in the quickest time possible. This is the default setting.

Configuring rebuilding of layer settings:

You can allow the rebuilding of the topology layers to display an accurate topology when you configure a partial discovery.

To rebuild the topology layers following a partial discovery, select the **Enable Rediscovery Rebuild Layers** setting on the **Advanced** tab within the Configuration option in the Discovery Configuration GUI. If you specify that topology layers *should* be rebuilt following partial discovery, the result is an accurate topology showing all connectivity. However, the process of adding new devices takes longer.

Related concepts:

"Option to rebuild topology layers" on page 301
You can specify whether to rebuild the topology layers following a partial rediscovery. Using this option, you can increase the speed of partial rediscovery.

Enabling discovery of remote neighbors for partial discovery:

You can improve the accuracy of connections found during a partial discovery by enabling the discovery of remote neighbors.

By default, remote neighbor discovery is off. Enabling remote neighbor discovery makes the discovery take longer.

With remote neighbour discovery on, Network Manager checks, during a partial discovery, whether any connections to remote neighbors have changed. (Remote neighbors in this context are connected devices that were in scope of the last full discovery but are out of scope of the current partial discovery.)

If the connections have changed, the connected devices are included in the partial discovery, resulting in a more accurate topology.

Restriction: If a connection between devices has changed, but the information about the connection is stored only on the device that is out of scope, then the change is not registered and the connected devices are not included in the partial discovery. Enabling remote neighbor discovery increases the accuracy of the topology, if it has changed, but does not ensure that all changes are discovered. For the most accurate topology, run a full discovery.

To enable remote neighbor discovery, select **Enable Rediscovery of Related Devices** on the Advanced tab within the Configuration option in the Discovery Configuration GUI.

Starting partial discovery from the GUI

Starting a partial discovery involves defining a seed and scopes.

If a full discovery has not been run since the last time that the discovery engine, **ncp_disco**, was started, you cannot start a partial discovery.

You can start a partial discovery of a device or subnet from the Active Discovery Status window. You can also discover specific devices by right-clicking them from within the Hop View and Network Views.

To start a partial discovery from the Active Discovery Status window, complete the following tasks.

1. Select the domain in which you want to run a discovery from the **Domain** menu. You can start to type the name of the domain, and matching domains are listed below the **Domain** field.
2. Click the downward-facing arrow next to the **Start Discovery** button ▷ ▾ and select **Start Partial Discovery** from the menu. The Partial Discovery window is displayed. Specify the IP addresses and subnets that contain the devices to be discovered
3. Under **Partial Discovery**, select the required nodes and subnets.
4. To add a new subnet or node, click **New.**
5. Complete the fields as follows and click **OK**:

   Rediscover
   Select one of the following options:

   IP Address
   Type the required IP address.

Subnet

Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

6. To add new scope zones, click **Scope**.

   Note: If you add a scope zone that is not included within the scope of the last full discovery, connections between devices in the new scope and the old scope might not be accurate until the next full discovery. Enabling remote neighbor discovery can help improve the accuracy of these connections.

7. To add a new discovery scope zone click **New** . To edit an existing scope zone, click the required entry in the list.
8. Complete the fields as follows and click **OK**:

Scope By:

Select one of the following options:

Subnet

Type the required subnet and specify the number of netmask bits. The **Netmask** field is automatically updated.

You can specify a subnet or an individual IP address using these fields.

- For example, to specify a Class C subnet 10.30.2.0, type `10.30.2.0/24`, where `10.30.2.0` is the subnet prefix, and `24` is the subnet mask.
- To specify an individual device, type an IP address and a subnet mask of `32`. For example, type `10.30.1.20/32`.

Wildcard

Use an asterisk (*) as a wildcard.

For example, to specify a scope of all IP addresses that begin with the 10.30.200. subnet prefix, type `10.30.200.*`.

Restriction: Network Manager does not support the IPv4–mapped IPv6 format and expects all IPv6 addresses to be in standard colon-separated IPv6 format. For example, Network Manager does not support an IPv4–mapped IPv6 address such as `::ffff:192.0.2.128`. Instead enter this address as `::ffff:c000:280` (standard colon-separated IPv6 format).

Protocol

Select the required Internet protocol: IPv4 or IPv6.

Action

Define the subnet range as an inclusion zone or exclusion zone. If the subnet range is an inclusion zone that you intend to ping during the discovery, click **Add to Ping Seed List**. Clicking this option automatically adds the devices in the scope zone as a discovery seed devices.

Restriction: The `Add to Ping Seed List` option is not available for IPv6 scope zones. This prevents ping sweeping of IPv6 subnets, which can potentially contain billions of devices to be pinged. Ping sweeping of IPv6 subnets can therefore result in a non-terminating discovery.

9. Click **OK** then click **Go**. When a partial discovery is running, the **Start Discovery** button is toggled off ▷.

Related concepts:

"About types of discovery" on page 1
Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Related tasks:

"Starting a discovery" on page 43
After you configure a discovery, you can start and, if necessary, stop the discovery.

"Monitoring network discovery from the GUI" on page 131
From the Active Discovery Status page, you can monitor the status and progress of the current discovery, investigate the work of the discovery agents, and view details of the last discovery.

Manually discovering a device or subnet from the command line

You can manually discover a device or subnet from the command line.

To manually discover a device or subnet from the command line, make inserts to the `finders.rediscovery` table using `ncp_oql`, specifying the IP address or subnet to be discovered, as described in the following example.

Manual discovery

To manually discover the device with IP address 192.168.1.2, first start the OQL Service Provider using the following command:

```
ncp_oql -domain NCOMS -service Disco
```

Having logged into the OQL provider, run the following query (note that the command is entered on one line):

```
insert into finders.rediscovery (m_Address, m_RequestType) values
("192.168.1.2", 1);
```

When discovery of a device is forced like this, `ncp_disco` immediately passes it to the Ping finder to confirm it exists, and, if it does, triggers the appropriate agents to reanalyze it. If the connections from the device have changed, neighboring devices might also be discovered.

Removing a device from the network

You can manually remove a device that is known to be scheduled for permanent removal from the network.

1. Suspend polling against the device. This prevents any spurious alerts being raised against the device by the monitoring system as the device is powered off.
2. Physically remove the device from the network.
3. Immediately before the next full network discovery, set the linger time for the record of the device in ncp_model to 0.

Related tasks:

"Manually discovering a device or subnet" on page 156
You can manually discover devices so that the network topology in Network Manager matches the network.

Setting the linger time for a device

The value of the LingerTime field indicates how many discoveries a device can fail to be found in before it is assumed to have been removed from the network and its record is removed from the topology. Setting the LingerTime field to zero ensures that when the device is not found in the next discovery, its record is immediately removed from the topology.

To set the LingerTime field to zero:

1. Enter a command similar to the following to start the OQL Service Provider:
   ```
   ncp_oql -domain NCOMS -service Model
   ```
2. Update the LingerTime field in the master.entityByName table for all entities that represent the device. For example, if the device is called `core-router.abcd.com`, enter the following command, on one line:
   ```
   update master.entityByName set LingerTime = 0
   where EntityName like 'core-router.abcd.com';
   ```

Manually updating device details

Updated device details are sometimes not detected by discovery.

Sometimes changes you make to a device, such as giving it a new name, are not detected by a subsequent discovery. When this happens, you can use the Remove Node tool to remove the device from the network topology, and then rediscover the device based on its new details.

To manually update a device, follow these steps:

1. Run the `RemoveNode.pl` script against the device.
2. Rediscover the device.

Chapter 6. Troubleshooting discovery

You can troubleshoot discovery by monitoring discovery events and by running discovery reports. You can also configure your own discovery events.

Troubleshooting discovery with reports

The troubleshooting reports provide easy visibility into the discovery results to help with verification and troubleshooting of both the discovery results and the network itself.

Network Manager uses the Tivoli® Common Reporting component to generate reports. More information on Tivoli Common Reporting is located at

- Tivoli Common Reporting Information Center
- developerWorks® Tivoli Common Reporting

To access the reports in the Network Manager GUI, click **Reporting** > **Common Reporting** in the navigation pane.

You can use reports for verifying and troubleshooting discovery results such as the examples in Table 22.

For more information on the Network Manager reports, refer to the *IBM Tivoli Network Manager IP Edition Administration Guide*.

Table 22. Report categories to use for discovery troubleshooting

Troubleshooting task	**Consult this report category and report**	**Benefit of the report**
Identifying all discovered nodes and interfaces	Utility reports: Discovered nodes and interfaces flat file list	This report lists all nodes and interfaces that were discovered. It also marks interfaces or ports connected to network devices. It allows you to check that specific devices and interfaces were actually discovered.
Resolving mismatches	Troubleshooting reports: Connected Interface Duplex Mismatch	This report provides a list of connections that have mismatches between half- and full-duplex devices, where one end of the connection is half-duplex and the other end is full-duplex. This mismatch is one of the key configuration problems that network managers have to find to resolve performance or availability issues.
Resolving inaccessible devices	Troubleshooting reports: Devices with no SNMP Access	This report identifies the devices that do not have SNMP access. You can then identify the reason for the SNMP access failure.
Resolving unconnected devices	Troubleshooting reports: Devices With No Connections	This report lists unconnected devices as a first step in determining why the discovery found no network connections for a device.

Table 22. Report categories to use for discovery troubleshooting (continued)

Troubleshooting task	Consult this report category and report	Benefit of the report
Resolving unclassified devices	Troubleshooting reports: Devices with Unclassified SNMP Object IDs	Using these reports, you can create leaf-node AOC files for the new device class.
	Asset reports: • Interface Availability • Summary By Device Class • Vendor and Device Availability	
Resolving devices with unregistered SNMP object IDs	Troubleshooting reports: Devices with Unknown SNMP Object IDs	Using the information in this report, you can modify the AOC files associated with the unregistered devices.
Identifying devices pending deletion	Troubleshooting reports: Devices Pending Delete on Next Discovery	This report displays information on devices to be deleted from the topology if they are not found during the next discovery cycle. The report allows you to check that removal of devices from the topology is progressing, and to identify devices erroneously scheduled for removal.

Monitoring discovery status

You can view discovery status messages to understand the status and progress of the discovery. You can also configure your own discovery events.

Process flow for creating discovery events

Discovery events are created during the discovery process showing the progress of agents, stitchers, and finders. These events are sent to and stored in Tivoli Netcool/OMNIbus and can be viewed using the Web GUI.

Discovery events are created in the following stages:

- During the data collection phase of discovery, dedicated stitchers (AgentStatus and FinderStatus) detect whether finders and agents have started or stopped.
- During the data processing phase, a dedicated stitcher (CreateStchTimeEvent) detects key events; for example, discovery has started building the working entities table, or the containment table.
- Whenever one of the above stitchers detects an event, it writes that event to the `disco.events` table.
- The DiscoEventProcessing stitcher responds to an insert into the `disco.events` table and creates and sends the appropriate event to the probe for Tivoli Netcool/OMNIbus, nco_p_ncpmonitor, which then forwards the event to the ObjectServer.
- You can switch the generation of discovery events on or off by setting the value of the `m_CreateStchrEvents` field in the `disco.config` table.

You can configure your own discovery events by writing a stitcher to detect the desired event and write that event data to the `disco.events` table.

Related reference:

"disco.events table" on page 197
The events table constrains discovery events generated to a standard format. An event is generated by inserting a record into this table.

"Main discovery stitchers" on page 341
This topic lists all discovery stitchers.

Monitoring discovery status messages

You can view discovery status messages to understand the status and progress of the discovery.

The discovery processes, including agents, stitchers, and finders, send messages to IBM Tivoli Netcool/OMNIbus when they start and stop. You can view these messages to see if the discovery processes are running as expected, and to gauge the overall progress of discovery.

To view discovery process status messages, complete the following tasks.

1. Click **Availability** > **Events** > **Active Event List (AEL)** to view the **Active Event List (AEL)**.
2. Apply a filter to the AEL so that only events with an `Agent` of `ncp_disco` are displayed.
3. Optional: Refine the filter or sort on **EventId** to view only specific kinds of discovery events.
4. Ensure that the **LocalPriObj** and **LocalSecObj** columns are displayed in the AEL. These columns contain information for discovery events. (Not all columns are used by all events.)

Troubleshooting discovery agents

You can use the Discovery Status GUI to troubleshoot discovery problems associated with discovery agents.

Troubleshooting an unusually long discovery

A discovery might be taking a long time to complete because an agent is unable to complete processing on a specific device. Use the **Agents Status** section to determine which agent is taking a long time to complete and which device it is working on.

To use the **Agents Status** section to determine if the cause of the problem is an agent that is blocked on a device, complete the following steps:

1. Open the **Agents Status** section by clicking **Discovery** > **Network Discovery Status**, and then clicking the **Agents Status** tab.
2. Set the Phases drop-down list above the upper Agents table to Interrogating Devices. The upper Agents table now displays only agents that are scheduled to complete in the first discovery phase, Interrogating Devices.

   Note: This problem usually occurs during the first discovery phase, Interrogating Devices.
3. Ensure that the **State** column is sorted in descending order. The agents appear by default in descending order of agent state, as listed in the following table.

Table 23. Agent states

State	Value	Icon	Description
Died	5	⊗	The agent has terminated unexpectedly. This is a potential discovery problem.
Finished	4	✓	The agent is still running but has finished processing of all the IP addresses in its queue. The agent is still available to process any further agents placed in the queue.
Running	3	✻	The agent is currently processing IP addresses.
Starting	2	✻	The agent is starting up.
Not running	1	■	The Agent is not running.

4. Scroll down the table to find the agents that have the status Running ✻. These are the agents that are still processing devices. If the discovery has been running for an unusually long time, then there might be just one agent that still has Running status ✻. This is the blocked agent.
5. Select one of the agents with Running status ✻. By default, the lower table now displays all the IP addresses that are still queued for this agent.
6. Click the **All** radio button above the lower table. The lower table now shows all IP addresses that have been processed by this agent, that are still being processed by this agents, or that are in the agent queue.
7. Ensure that the **State** column is sorted in descending order. The IP addresses appear by default in descending order of agent state, as listed in the following table.

Table 24. IP address states

State	Value	Icon	Description
Died	5	⊗	Processing of the IP address terminated unexpectedly. This is a potential discovery problem.
Finished	4	✓	An agent has completed processing this IP address.
Running	3	✻	An agent is currently processing this IP address.
Starting	2	✻	An agent is beginning to process this IP address.
Not running	1	■	This IP address is not currently being processed.

8. Scroll down the table to find the IP addresses that have the status Running ✻. These are IP addresses that are still being processed by this agent. If the agent is stuck on a single device, then there will only be one IP address with Running status ✻.
9. Look at the other information in the table to find out more about this IP address. The Elapsed Time column indicates how long the agent has been processing this device. The SNMP Access column indicates whether the agent was able to gain SNMP access to this device.

If the agent was unable to gain SNMP access to the device, there might be a problem with SNMP community string settings. Further investigation of this device is required.

Related tasks:

"Monitoring discovery agent progress" on page 134
You can use the **Agents Status** section to monitor the progress of the discovery agents through each of the discovery phases.

Identifying failed agents

A source of discovery failure can be agents that terminate unexpectedly during discovery. Use the **Agents Status** section to determine if any agents have terminated unexpectedly.

To use the **Agents Status** section to determine if any discovery agents are not running correctly, complete the following steps:

1. Open the **Agents Status** section by clicking **Discovery** > **Network Discovery Status**, and then clicking the **Agents Status** tab.
2. Ensure that the Phases drop-down list above the upper Agents table is set to All Phases. The upper Agents table now displays all agents that have started so far in this discovery.
3. Click the **State** column in the upper Agents table so that the agents are ordered in descending order of **State**. The agents now appear in the table in alphabetical order of status.
4. Any agents that have terminated unexpectedly will be at the top of the table and will have the status Died.

Further investigation is required to determine why this agent terminated unexpectedly.

Related tasks:

"Monitoring discovery agent progress" on page 134
You can use the **Agents Status** section to monitor the progress of the discovery agents through each of the discovery phases.

Troubleshooting missing devices

If a device that you expect to find in your network topology is not present, follow these steps to troubleshoot the problem.

Before following these steps, run a full discovery with feedback enabled.

To check some common causes for a device not being found in the network maps, complete the following steps.

1. Verify that the device you're looking for is running and connected to the network.
2. Search for the device.
   a. Search for the device in the network maps by hostname and then by IP address.
   b. If you know which devices it is connected to, try finding one of the connected devices in the Network Hop View. Then set the number of hops to 1 and see if the device is shown as connected.
3. Check whether the device is in scope. Review the discovery scope, including exclusion zones, in the **Scope** tab of the Network Discovery Configuration GUI.
4. Check if the device is being filtered out of the discovery.
   a. Click **Filters**.

b. Review the prediscovery and postdiscovery filters to ensure that the device is not being prevented from being discovered or instantiated.

Related tasks:

"Setting discovery filters" on page 28
Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Troubleshooting an idle discovery

If you start the discovery, and after some minutes no devices have been discovered, follow these troubleshooting steps.

If the discovery status stays in phase zero (idle) after you start it, and no devices have been discovered, try the following troubleshooting steps.

1. If you are using the file finder, check that you have correctly specified which field in the seed file contains the IP address and which contains the host name. You can verify these settings in the Discovery Configuration GUI.
2. If you are using the ping finder and pinging individual IP addresses, check that these IP addresses are reachable. If not, you might have a network outage or a firewall issue.
3. Verify that the seed IP addresses are in scope. Even if you add an address to the ping finder or file finder, the device is not pinged or instantiated if it is not included in the scope. For example, if your discovery scope is 172.16.1.0 /24 and your seeds are in the 192.168.1.0 /24 network, then the finders cannot find them.
4. If you are pinging a large, sparsely populated subnet, for example, a class B subnet containing only 10 devices, the ping finder might take a long time to find the first device.

If you need to review the discovery logs, refer to the information about locating log files and changing logging levels in the *IBM Tivoli Network Manager IP Edition Administration Guide*.

Related tasks:

"Starting a discovery" on page 43
After you configure a discovery, you can start and, if necessary, stop the discovery.

Removing discovery cache files

Remove discovery cache files to perform a new, clean discovery.

To remove the current network discovery for a domain, you must remove all discovery cache files. You might want to do this when you need to remove all data from a previous discovery, or when directed to do so by IBM Support.

This procedure deletes all current discovery cache files and clears the discovery database, effectively resetting discovery. After performing this procedure, you must run a new full discovery of your network.

Note: Because the network topology is stored separately in the NCIM database, this procedure does not remove your network maps. However, any changes made to your network since the last discovery are reflected in the next discovery.

Perform the following procedure to remove all discovery cache files:

1. Stop all Network Manager processes using the `itnm_stop` script.
2. Navigate to the `$NCHOME/var/precision` directory and remove all files that belong to the domain you wish to remove. Files that belong to a particular domain have the domain in the file name. For example, a configuration file belonging to the domain NCOMS would be called `file_name.NCOMS.cfg`.
3. Optional: You can archive or remove existing log files and start the new discovery with fresh log files. The log files for the following processes are relevant:
   - **ncp_disco**
   - **ncp_df_***
   - **ncp_agent***
   - **ncp_disco_perl_agent***
4. Restart the Network Manager processes using the `itnm_start` script.

   New, empty log files are automatically created when the Network Manager processes are restarted using the `itnm_start` scripts.
5. Perform a new network discovery.

Troubleshooting illegal characters in the Informix database

If you see an error message about illegal characters in insert statements into the Informix database, follow these steps to troubleshoot the problem.

If you have network devices with characters in their descriptions that are not allowed in the locale set in the Informix database, you might see an error message similar to the following:

```
Warning: W-RIV-002-206: [4115626896t] CMdlDbEntityMgr.cc(647)
A database 'execute' operation has failed :
SQLRETURN = -1 CNcpODBCSth.cc line 233 : [Informix][Informix ODBC Driver][Informix]
An illegal character has been found in the statement.
```

1. Back up and edit the `SnmpStackSchema.cfg` file.
2. Locate the line that configures an insert into the `snmpStack.conversionCfg` table and edit it to the following:

   ```
   insert into snmpStack.conversionCfg values (1);
   ```
3. Save and close the file.

The SNMP Helper substitutes characters returned from devices that are not allowed in the Informix database locale with the question mark character: '?'.

The SNMP Helper substitutes characters on only those objects that are configured in the snmpStack.multibyteObjects table.

Chapter 7. Enriching the topology

You can enrich the topology by adding extra context to the information discovered by the discovery process. For example, you can add custom tags to devices to show the customer, location, or other information associated with that device. You can then use this custom information to visualize or poll your network.

This section presents different examples of topology enrichment. Use these examples for an idea of the different ways in which the topology can be enriched and the different methods available to enrich your topology.

Adding tags to entities

You can associate one or more name-value pair tags with discovered entities.

The following table shows an example of a device with IP address 172.20.3.20, with two associated name-value pair tags.

Table 25. Example of name-value pair tags

IP Address	Name	Value
172.20.3.20	customer	acme
172.20.3.20	location	london

Once the discovery has tagged your IP addresses with custom name-value pair tags, you can use the custom name-value pair information to perform custom visualization and polling tasks. For example, you could create a custom network view to display all IP addresses that are tagged with the location "london".

Customizing the discovery

Use one of the following ways to customize the discovery to add name-value pair tags to discovered entities: using the file finder, or using custom tag tables. If you use the custom tag tables then you can also use logic defined in the GetCustomTag stitcher to evaluate the value of the add name-value pair tag.

Adding tags to entities using the File finder

If you are using the File finder to seed your discovery, then you can add name-value pair tags to entities by adding extra columns to the seed file read by the File finder.

The example procedure below assumes that you are adding the following extra columns to your File finder seed file:

- customer
- location

The following example text file fragment shows what the seed file might look like:

```
vi /var/tmp/logged_hosts

172.16.1.21       lnd-dharma-acme       acme        london
172.16.1.201      lnd-phoenix-acme      acme        london
172.16.1.25       prs-sun-acme          acme        paris
172.16.2.33       ranger1               telecorp    newyork
```

```
172.16.2.34       ranger2               telecorp    newyork
~
"/var/tmp/logged_hosts" [Read only] 4 lines, 190 characters
```

In this example text file fragment, the third column holds customer information, and the fourth column holds location information.

1. Edit the DiscoFileFinderParseRules.cfg configuration file.
2. In this configuration file, configure the File finder to parse the seed file using an insert similar to the example. Ensure that you configure the `m_ColDefs` field to match the new custom tag columns.

   ```
   insert into fileFinder.parseRules
   (
               m_FileName,
               m_Delimiter,
               m_ColDefs
   )
   values
   (
               "/var/tmp/logged_hosts",
               "[    ]",
               [
                   {
                       m_VarName="m_UniqueAddress",
                       m_ColNum=1
                   },
                   {
                       m_VarName="m_Name",
                       m_ColNum=2
                   }
                   {
                       m_VarName="m_CustomTags->customer",
                       m_ColNum=3
                   },
                   {
                       m_VarName="m_CustomTags->location",
                       m_ColNum=4
                   }
               ]
   );
   ```

   This insert instructs the File finder to do the following:
   - Parse `/var/tmp/logged_hosts`.
   - Treat white space as the data separator.
   - Use the following column definitions:
     - `m_UniqueAddress` for the first column
     - `m_Name` for the second column
     - `m_CustomTags->customer` for the third column
     - `m_CustomTags->location` for the fourth column
3. Edit the DbEntityDetails.cfg file and configure an insert similar to the following:

   ```
   insert into dbModel.entityDetails
   (
       EntityType,
       EntityDetails
   )
   values
   (
       1,  -- chassis
       {
           Customer = "eval(text, '&ExtraInfo->m_CustomTags->customer')",
   ```

```
            Location = "eval(text, '&ExtraInfo->m_CustomTags->location')"
        }
);
insert into dbModel.entityDetails
(
     EntityType,
     EntityDetails
)
values
(
     2,  -- port/interface
     {
         Customer = "eval(text, '&ExtraInfo->m_CustomTags->customer')",
         Location = "eval(text, '&ExtraInfo->m_CustomTags->location')"
     }
);
```

4. Restart Network Manager to propagate the configuration file changes:

```
itnm_start ncp -domain domain
```

Related tasks:

"Enabling polling and visualization using the custom tags" on page 178
After you have customized discovery to add custom tags you must ensure that the NCIM topology database entityDetails table is updated with the custom tags. This enables you to poll and visualize devices using these custom tags.

Adding tags to entities using custom tag tables

You can add name-value pair tags to entities by creating inserts containing the name-value pair data into the disco.ipCustomTags table or into the disco.filterCustomTags table.

Adding tags to entities using the disco.ipCustomTags table:

You can associate name-value pair tags to unique IP addresses using the disco.ipCustomTags table.

The example procedure below assumes that you are adding the following two custom name-value pair tags to entities in your discovery:

- customer
- location

This example uses the disco.ipCustomTags table to configure the following name-value pair tags:

Table 26. Example of name-value pair tags

IP Address	Name	Value
172.16.1.21	customer	acme
172.16.1.21	location	london
172.16.1.201	customer	acme
172.16.1.201	location	london
172.16.1.25	customer	acme
172.16.1.25	location	paris
172.16.2.33	customer	telecorp
172.16.2.33	location	newyork
172.16.2.34	customer	telecorp
172.16.2.34	location	newyork

1. Edit the DiscoConfig.cfg configuration file.
2. In this configuration file, add an insert similar to the following.

```
insert into disco.ipCustomTags
(
            m_UniqueAddress,
            m_CustomTags,
 )
values
(
            '172.16.1.21',
            {
                customer="acme",
                location="london"
            }
);

insert into disco.ipCustomTags
(
            m_UniqueAddress,
            m_CustomTags,
 )
values
(
            '172.16.1.201',
            {
                customer="acme",
                location="london"
            }
);

insert into disco.ipCustomTags
(
            m_UniqueAddress,
            m_CustomTags,
 )
values
(
            '172.16.1.25',
            {
                customer="acme",
                location="paris"
            }
);

insert into disco.ipCustomTags
(
            m_UniqueAddress,
            m_CustomTags,
 )
values
(
            '172.16.2.33',
            {
                customer="telecorp",
                location="newyork"
            }
);

insert into disco.ipCustomTags
(
            m_UniqueAddress,
            m_CustomTags,
 )
values
(
            '172.16.2.34',
            {
                customer="telecorp",
                location="newyork"
            }
);
```

3. Save the DiscoConfig.cfg configuration file.

You must now configure the DbEntityDetails.cfg configuration file to ensure that, following discovery, the NCIM topology database entityDetails table is updated with the custom tags.

Related tasks:

"Enabling polling and visualization using the custom tags" on page 178
After you have customized discovery to add custom tags you must ensure that the NCIM topology database entityDetails table is updated with the custom tags. This enables you to poll and visualize devices using these custom tags.

Adding tags to entities using the disco.filterCustomTags table:

You can associate name-value pair tags to a filtered set of IP addresses using the disco.ipCustomTags table.

You can filter IP addresses based on a wide variety of criteria. For example, you can filter based on device name, based on IP address, or based on VLAN identifier. The example procedure below applies a filter based on IP address, and uses the disco.filterCustomTags table to configure the following name-value pair tags to all IP addresses in the subnet 172.20.3.0/24:

Table 27. Example of name-value pair tags

IP Address	Name	Value
172.20.3.0/24	customer	acme
172.20.3.0/24	location	london

1. Edit the DiscoConfig.cfg configuration file.
2. In this configuration file, add the following insert:

```
insert into disco.filterCustomTags
(
            m_Filter,
            m_CustomTags,
 )
values
(
            "m_UniqueAddress LIKE '172.20.3'",
            {
                customer="acme",
                location="london"
            }
);
```

3. Save the DiscoConfig.cfg configuration file.

Other examples of filters

The procedure above applies a filter based on IP address: `"m_UniqueAddress LIKE '172.20.3'"`.

You can create a filter based on any attributes associated with discovered entities. For example, you could apply the following filters:

- Filter based on entity name: "m_Name LIKE 'lon'"
- Filter based on VLAN identifier of a VLAN entity: "m_LocalNbr->m_VlanID = 102"

You must now configure the DbEntityDetails.cfg configuration file to ensure that, following discovery, the NCIM topology database entityDetails table is updated with the custom tags.

Related tasks:

"Enabling polling and visualization using the custom tags" on page 178
After you have customized discovery to add custom tags you must ensure that the NCIM topology database entityDetails table is updated with the custom tags. This enables you to poll and visualize devices using these custom tags.

Enriching the topology using the GetCustomTag stitcher:

You can use the GetCustomTag stitcher to use logic to evaluate the value part of the name-value pair.

The example procedure below makes use of the default logic in the GetCustomTag.stch stitcher to add the following custom name-value pair tag to all IP addresses in the subnet 172.20.3.0/24:

Table 28. Example of name-value pair tags

IP Address	Name	Value
172.20.3.0/24	Customer	A-Z Inc., London

1. Edit the DiscoConfig.cfg configuration file.
2. In this configuration file, add the following insert:

```
insert into disco.filterCustomTags
(
            m_Filter,
            m_StitcherTagName,
 )
values
(
            "m_UniqueAddress LIKE '172.20.3'",
            'Customer'
);
```

   This insert configures the system to perform the following action for each IP address in the subnet 172.20.3.0/24: call the GetCustomTag.stch stitcher and pass the name part of the Customer tag to this stitcher. The GetCustomTag.stch stitcher will then evaluate the value for the Customer tag.
3. Save the DiscoConfig.cfg configuration file.

You must now configure the DbEntityDetails.cfg configuration file to ensure that, following discovery, the NCIM topology database entityDetails table is updated with the custom tags.

Related tasks:

"Enabling polling and visualization using the custom tags" on page 178
After you have customized discovery to add custom tags you must ensure that the NCIM topology database entityDetails table is updated with the custom tags. This enables you to poll and visualize devices using these custom tags.

Example: GetCustomTag.stch stitcher:

Use this topic to understand how the GetCustomTag.stch stitcher works.

AddCustomTags.stch stitcher

The GetCustomTag.stch stitcher is called by the AddCustomTags.stch stitcher.

The AddCustomTags.stch stitcher loops through the tags and the entities in the disco.ipCustomTags and disco.filterCustomTags tables. If, in either of these tables, the `m_StitcherTagName` field is set, then the AddCustomTags.stch stitcher calls the GetCustomTag.stch stitcher and passes the relevant entity name and the `m_StitcherTagName` field as parameters. The `m_StitcherTagName` field holds the name part of a name-value pair tag; for example, this field might have the value of 'Customer'. Once the AddCustomTags.stch stitcher has constructed all name-value pairs for the IP addresses defined in the disco.ipCustomTags and disco.filterCustomTags tables, it then passes the information downstream.

Note: The AddCustomTags.stch stitcher retrieves the entity name by performing a lookup in the workingEntities.finalEntity table using the IP address information provided in disco.ipCustomTags or disco.filterCustomTags table.

GetCustomTag.stch stitcher

The GetCustomTag.stch stitcher takes as input a single entity name and a the `m_StitcherTagName` field, and uses logic to evaluate the value part of the name-value pair. By default the stitcher contains the code described here. You can customize this stitcher to work with different name-value pairs and you can change the logic defining how the value is calculated.

Table 29. Line-by-line description of the GetCustomTag.stch stitcher

Line numbers	Description
15	Set the value of the entityName variable from the first argument received from the AddCustomTags.stch stitcher. The entityName variable holds the entity name associated with the IP address for which the stitcher is evaluating the value of a name-value pair tag.
16	Set the value of the tagName variable from the first argument received from the AddCustomTags.stch stitcher. This is the name of the tag for which the value is to be evaluated.
18	Set the value variable to zero. The value variable will be returned by the stitcher and will hold the evaluated value of the name-value pair tag.
20-29	If the name of the tag to be evaluated is 'Customer', then calculate the value of the tag. Calculate the value in the following way: if the entity name contains the text pattern `lon` then set the value variable to the customer name "A-Z Inc. London".
31	Return the value of the tag.

```
//
// This stitcher retrieves the value for a custom tag name
//
UserDefinedStitcher
{
 StitcherTrigger
 {
  //
  // Called from another stitcher
  //
 }

 StitcherRules
 {
  text entityName = eval(text,'$ARG_1');
  text tagName = eval(text,'$ARG_2');

  text value = NULL;

  if(tagName == "Customer")
  {
   // insert logic to retrieve custom tag
   // As an example, we extract the hostname part of the name
   int count = MatchPattern(entityName, '(lon)');
   if (count == 1)
   {
    value = "A-Z Inc., London";
   }
  }

  SetReturnValue(value);
 }
}
```

Enabling polling and visualization using the custom tags

After you have customized discovery to add custom tags you must ensure that the NCIM topology database entityDetails table is updated with the custom tags. This enables you to poll and visualize devices using these custom tags.

To enable polling and visualization based on custom tags:

1. Go to the `$NCHOME/etc/precision` directory and edit the `DbEntityDetails.cfg` file.
2. Uncomment the **`insert`** statement. For an example of the **`insert`** statement, see "Sample insert statement" on page 179.

MODEL checks the ExtraInfo section of each interface record for the following fields:

- m_CustomerName
- m_CustomerType

If either field is found, the value is inserted into the NCIM topology database entityDetails table and is associated with an entityId that is equal to the value specified in the current MODEL interface record. For more information on the entityDetails table, see the *IBM Tivoli Network Manager IP Edition Topology Database Reference*.

If a MODEL interface record does not contain an m_CustomerType or an m_CustomerName attribute in the ExtraInfo section, or if either of these fields has a NULL value, no row is added to the entityDetails table for that interface record.

Sample insert statement

```
/////////////////////////////////////////////////////////
//
// This file provides a means to extend the NCIM database
// schema by adding key-value pair data to the database
// table named entityDetails.
//
//
//
// The following example assumes that a custom stitcher has been created
// with the ability to populate the ExtraInfo section of chassis
// entities with the name and type of each customer.
//
//  insert into dbModel.entityDetails
//  (
//      EntityType,
//      EntityDetails
//  )
//  values
//  (
//      1,  -- chassis
//      {
//          CustomerName = "eval(text, '&ExtraInfo->m_CustomerName')",
//          CustomerType = "eval(text, '&ExtraInfo->m_CustomerType')"
//      }
//  );
```

You can now run a full discovery to discover your network with the custom tags.

Visualizing the enriched topology

Once you have created a topology in which certain entities are tagged with one or more name-value pairs, you can create a custom network view to display the tagged entities. You can also use the Network Hop View to search for tagged entities.

In this example, you create a distinct dynamic network view that categorizes devices by customer. This example assumes that you have tagged IP addresses with a single name-value pair containing the customer name associated with that IP address. The GetCustomTag.stch stitcher contains an example of how to do this.

For more information on creating network views, see the *IBM Tivoli Network Manager IP Edition Network Visualization Setup Guide*.

1. Click **Availability** > **Network Availability** > **Network Views**. Click **New View** .
2. Complete the **General** tab as follows:

   Name Type a name for the network view, dynamic view, or network view container.

   Important: It is best practice to use network view names containing Latin characters only. Network views names containing non-Latin characters (for example Cyrillic characters) are not supported as they cannot be imported and exported when migrating to a new version of Network Manager.

   Parent Select the node under which the view appears in the hierarchy in the Navigation Tree. To display the view on the top level, select `NONE`.

   Type Select `Dynamic Views – Distinct`.

Layout
Select `Orthogonal`, `Circular`, `Symmetric`, `Hierarchical`, or `Tabular` layout.

Map Icon
If you want a different icon than the default cloud icon to represent the view, click **Browse** ... to browse for an icon.

Tree Icon
If you want a different icon than the default cloud icon to represent the view, click **Browse** ... to browse for an icon.

Background Image
Click **Browse** ... to browse for an image to use as the background for the view.

Background Style
Specify whether the background image is to be centered or tiled.

Line Status
Specify how the lines that represent the links between devices should be rendered.

You can choose not to display any status, or to display the system default. Alternatively, lines can be colored based on the associated AEL event with the highest severity, and can appear with an additional severity icon.

3. Click the **Filter** tab. From the **Domain** list, select your network domain.
4. In the **Fields** list, select the topology database table and field that correspond to the categories and subcategories that you want to define.
   a. Click **Add...**.
   b. From the **Table** list, select the `entityDetails` database table. The **Field** list is automatically populated based on your selection.
   c. Select the `keyName` field from the **Field** list.
   d. Click **OK**.

   As you select fields, the **Preview** list is updated to show the relationships between the categories that you selected.
5. From the **End nodes** list, specify whether you want end nodes, such as printers and workstations, to be displayed in the view.
6. From the **Connectivity** list, select the required connectivity:

Option	Description
`IP Subnets`	Displays device membership by subnet. To simplify the view and make subnet membership clear, this type of connectivity does not show all connections.
`Layer 2`	Displays all datalink connections. No logical connections are displayed.

Option	Description
Layer 3	Displays all logical connections. Routers are displayed. Switches are not displayed, unless they have an active connection that involves a layer 3 interface. Connections between layer 3 devices are displayed. Connections between a layer 3 and a layer 2 interface are displayed between the layer 3 interface and the subnet to which the layer 2 interface belongs.
OSPF	Displays connections based on discovered OSPF information that includes router roles, area membership, and connectivity.
PIM	Displays connections based on PIM adjacency information.
IPMRoute	Displays connections based on IP Multicast upstream and downstream routing information.
No connections	Does not present any of the discovered connections for the nodes shown in the view.

7. Click **OK**. The new view is added to the navigation tree in the Navigation Panel. If you added the view to a container, expand the container node to see the new view in the tree.

Now that you have created a distinct dynamic network view that categorizes devices by customer, you can create a poll policy that polls some or all of the customers.

Related tasks:

"Enriching the topology using the GetCustomTag stitcher" on page 176
You can use the GetCustomTag stitcher to use logic to evaluate the value part of the name-value pair.

Polling the enriched topology

Now that you have created a distinct dynamic network view that categorizes devices by customer, you can create a poll policy that polls some or all of the customers.

In this example, you use the Poll Policy Wizard to guide you through the creation of a poll policy. During the procedure you are provided the option to specify which network views to poll. By selecting the distinct dynamic network views that you created based on the customer name-value tag pairs, you are able to poll devices based on the customer name associated with those devices.

If you require a fully-featured poll policy with multiple poll definitions and full scoping features, then use the Poll Policy Editor. For more information on creating poll policies, see the *IBM Tivoli Network Manager IP Edition Event Management Guide*.

1. Click **Administration** > **Network** > **Network Polling**.
2. Click **Launch Poll Configuration Wizard** ✏ .
3. Click **Next**. Complete the Poll Policy Details page as follows:

Name Specify a name for the poll policy. Only alphanumeric characters, spaces and underscores are allowed.

Interval
Specify the required interval in seconds between poll operations. Click the arrows to change the value.

Poll Enabled
Specify whether the poll should be enabled. The poll is enabled by default. To disable the poll, clear this check box.

Store Poll Data
Select this check box to store the poll data so that it can be subsequently retrieved for reporting. The data is stored in the ncpolldata database.

Restriction: Storage of polled data is not supported for the Cisco Remote Ping, the Juniper Remote Ping, and the Generic Threshold poll definitions.

Definition
Select a poll definition from the list.

4. Click **Next**. On the Network Views page, navigate in the network views tree to the node containing the distinct dynamic network views that you created. Open the node and select the customer devices that you want to poll using this poll policy.
5. Click **Next**. On the Poll Policy Summary page, review the information that you specified and click **Finish**.

Appendix A. Discovery databases

There are various specialized databases that are used by ncp_disco, the component that discovers network device existence and connectivity, and by ncp_model, the component that manages, stores, and distributes the discovered network topology.

The ncp_disco component and ncp_model component store configuration, management, and operational information in databases. You can interrogate these databases by logging into the DISCO or MODEL service using the OQL Service Provider.

The ncp_disco databases can either be active or passive. When data is inserted into an active database, an action is automatically triggered; for example, another table is populated with data, or a script or stitcher is launched.

Related concepts:

"Filters" on page 5
Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

Related tasks:

"Setting discovery filters" on page 28
Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Discovery engine database

Use the Discovery engine (ncp_disco) database to configure the general options for the discovery process, and to track the discovery process.

The Discovery engine database, disco, is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table names are: disco.config; disco.managedProcesses; disco.status; disco.agents; disco.NATStatus.

disco.config table

The config table configures the general operation of the discovery process.

Table 30. disco.config database table schema

Column name	Constraints	Data type	Description
`m_NothngFndPeriod`		`Float`	The maximum time lapse, in seconds, between the discovery of one device and the discovery of the next device in the device discovery phase.
`m_PendingPerCent`		`Integer`	The maximum allowed ratio of pending devices to processing devices. A breach of this threshold condition instigates a full discovery (rather than a partial rediscovery).

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_CycleLimit		Integer	The number of discovery cycles to complete before instigating a full rediscovery (used by the FinalPhase stitcher).
m_RestartAgents		Integer	A flag that determines whether DISCO attempts to restart discovery agents that fail during their operation.
m_RestartFinders		Integer	Flag to determine whether to restart a failed finder.
m_DirScanIntvl		Integer	The time period between scans for modifications to the stitcher and agent files. If changes are found, the stitcher and agent definitions are loaded and the appropriate changes are made to the stitchers and agents.
m_WriteTablesToCache	Externally defined Boolean data type	Boolean Integer	Flag indicating whether to write a cache of the Discovery engine, ncp_disco, tables to disk. **Note:** Setting this flag results in discoveries that are slower than a standard discovery. • 1: Write cache of ncp_disco tables to disk. The tables that are defined in the failover database are cached and ncp_disco can be restarted at any point. • 0: Do not write cache of ncp_disco tables to disk. No tables are cached during the discovery and ncp_disco ignores any existing cache files if it is restarted.
m_MinResidentSize		Integer	The minimum initial size of DISCO in kilobytes (KB). The maximum value that you can specify is 500 MB (512 000 KB). Specifying an initial value speeds up the discovery by allocating the memory of DISCO in one block.
m_UseContext		Boolean Integer	Flag indicating whether this is a context-sensitive discovery. • 1: Specifies a context-sensitive discovery. • 0: Specifies that this is not a context-sensitive discovery.
m_RebuildLayers	Externally defined Boolean data type	Boolean Integer	Flag indicating whether to rebuild topology layers after partial rediscovery. • 1: Rebuild the layers. After partial rediscovery, topology layers stitchers are run. Partial rediscovery takes longer but results in a complete topology. • 0: Do not rebuild the layers. After partial rediscovery, topology layers stitchers are not run. The result is a much quicker partial discovery; however, connectivity associated with the newly discovered device is not fully seen in the topology.

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_DiscoProfiling		Boolean Integer	Flag indicating whether to profile the discovery. • 1: Profile the discovery. • 0: Do not profile the discovery.
m_ModelVlans	Externally defined Boolean data type	Boolean Integer	Flag indicating whether to switch off VLAN modelling. 1: This setting switches *on* VLAN modelling. When you make this setting, the AddGlobalVlans, CreateTrunkConnections and AddVlanContainers stitchers *are* called. 0: This setting switches *off* VLAN modelling. When you make this setting, the AddGlobalVlans, CreateTrunkConnections and AddVlanContainers stitchers are *not* called.
m_DisplayMode		Integer	Specifies how the display label used for GUI network and network hop views should be populated for main nodes. • 0 - Use Entity Name (default) • 1 - Use SysName. This option is useful when it is not desirable to name entities by sysName in the database (see m_UseSysName) but it is desirable to have the entities displayed in the GUI views with a sysName.
m_RTBasedVPNs	Externally defined Boolean data type default = 0	Boolean Integer	Flag indicating which type of MPLS discovery to perform. • 1: Set this value to select route target (RT)-based MPLS discovery. In this type of discovery, no label data is required, so the discovery is faster. The MPLS core view consists of all MPLS-enabled devices. • 0: Set this value to select label switch path (LSP)-based MPLS discovery. In this type of discovery, label data is discovered as this data is required in order to trace LSPs. The MPLS core view shows provider edge (PE) routers and provider (P) routers retrieved by tracing the label switched paths within the VPNs in scope.
m_UseIfName	Externally defined Boolean data type default = 0	Boolean Integer	Flag indicating which naming strategy to use when building interfaces. • 1: This setting indicates that you want to use ifName or ifDescr to name the interfaces rather than their ifIndex, card or port information. • 0: This setting indicates that you want to use the default naming convention for any device interface: `baseName[<card>[<port>]`

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_UseSysName	Externally defined Boolean data type default = 0	Boolean Integer	Flag indicating which naming strategy to use when naming devices. • 1: This setting indicates that you want to name devices using the value of the SNMP sysName variable as the main source of naming information. The sysName variable must be set and must be unique within the network. • 0: This setting indicates that you do not want to name devices using the value of the SNMP sysName variable as the main source of naming information.
m_CheckFileFinderReturns	Externally defined Boolean data type default = 0	Boolean Integer	Flag indicating whether to use the Ping finder to check the devices specified in the flat file supplied to the File finder. • 1: This setting tells the Ping finder to check the devices specified in the flat file supplied to the File finder. This setting is recommended if you have reason to doubt that some of the devices specified in the flat file are still connected to the network. • 0: This setting indicates that you do not want to perform any checking of the devices specified in the flat file supplied to the File finder.
m_InferCEs	Externally defined Boolean data type default = 0	Boolean Integer	Flag indicating whether to infer the existence of customer-edge (CE) routers. When enabled, DISCO creates a CE router entity for each provider-edge (PE) router interface that is on a /30 subnet and does not have CE information from another source. • 1: This setting tells DISCO to infer the existence of CE routers. • 0: This setting tells DISCO not to infer the existence of CE routers.

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_FeedbackCtrl	Default = 0	Integer	Flag indicating whether to use the feedback mechanism during the discovery. The feedback mechanism allows any new IP addresses to be fed back into the discovery and thus increases the size of the discovered network. Devices that are fed back are pinged by the Ping finder. **Note:** For feedback to work the ping finder must be activated. • 0: Feedback is off for all discoveries and rediscoveries. This option provides speed but discovers only those devices specified to the finders, and hence provides an incomplete topology. However, this setting ensures that discoveries and rediscoveries complete in the quickest possible time. • 1: Feedback is on for full discoveries, full rediscoveries, and partial rediscoveries. All IP addresses are pinged. This option provides a complete topology in all situations but takes the longest. • 2: Feedback is on for full discoveries and full rediscoveries, this ensuring a complete topology in these cases. In the case of partial rediscoveries there is no feedback. This ensures that the partial rediscovery runs in the quickest time possible. This is the default setting.
m_AllowVirtual	Default = 0	Integer	Flag indicating whether to allow virtual IP addresses as part of the discovery. • 0: Do not perform any discovery for virtual IP addresses. • 1: Perform discovery for virtual IP addresses. This is the default setting. • 2: Perform discovery for virtual IP addresses only if the address is defined in the scope.special table. This table defines management IP addresses.
m_PingVerification	Default = 2	Integer	Option to check whether an interface is able to be pinged. If the device is not pingable, then Network Manager does not poll the device for alerts • 0: Do not check pingability: Network Manager performs no pingability check on any of the interfaces discovered. Interfaces will be polled regardless of whether they are pingable at discovery time. • 1: Check pingability: perform a pingability check, following discovery, for every interface discovered. • 2: Determine best method: sets the pollability flag for an interface based on whether feedback was active during the discovery..

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_CreateStchrEvents	Externally defined Boolean data type default = 1	Boolean Integer	Specifies whether to create discovery events to be sent to the ObjectServer. This field takes the following values: • 0: Do not generate discovery events. • 1: Generate discovery events.
m_RTVPNResolution		Integer	Specifies whether to apply fine control over Layer 3 VPN resolution and naming in a route target-based discovery: • 1: Use route target (default). • 2: Use VRF.
m_InferPEsUsingBGP		Boolean Integer	Specifies whether to infer the existence of provider-edge (PE) routers using BGP information on customer-edge (CE) routers: • 0: Do not infer PEs. • 1: Infer PEs.
m_BuildLogicalCollections		Boolean Integer	Specifies whether to build logical collection entities to group together items such as VTP domains, OSPF areas, and MPLS VPNs: • 0: Do not build logical collection entities. • 1: Build logical collection entities.
m_InferDumbHubs		Boolean Integer	Specifies whether to infer the existence of dumb hubs on your network: • 0: Do not infer the existence of dumb hubs. • 1: Infer the existence of dumb hubs.
m_RediscoverRelatedDevices		Boolean Integer	In a partial rediscovery when a device has changed, specifies whether to rediscover the related devices if the connection appears to have changed: • 0: Do not rediscover the related devices if the connection appears to have changed. • 1: Rediscover the related devices if the connection appears to have changed.
m_DiscoOnStartup		Boolean Integer	Specifies whether a discovery should automatically start when the Discovery engine, ncp_disco, is started: • 0: Do not automatically start a discovery. • 1: Automatically start a discovery.
m_FindersOnStartup		Boolean Integer	Specifies whether the finders should automatically start when the Discovery engine, ncp_disco, is started : • 0: Do not automatically start finders. • 1: Automatically start finders.

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_SubnetFiltering		Integer	Alters which interfaces are included in subnet-based connections: • 0: No filtering • 1: Filter out VRF interfaces (consider using m_VpnASTagging instead of this mode as it improves connectivity in all layers rather than just layer 3). • 2 - Filter out in-scope interfaces known to hold inaccessible duplicate IPs. • 3 - Automatic. Decides best approach based on other configuration options.
m_VerifyCDPUsingDeviceId		Boolean Integer	Specifies whether to verify the CDP links using the CDP device ID. Occasionally the CDP device ID has been found to be unreliable. Switching on m_VerifyCDPUsingDeviceId will improve CDP connectivity if the Device ID is accurate but might degrade connectivity if the Device ID is inaccurate. • 0: Do not verify the CDP links using the CDP device ID. • 1: Verify the CDP links using the CDP device ID.
m_UseIfIndex		Boolean Integer	Specifies whether to name interfaces using the ifIndex only. This setting overrides the m_UseIfName setting. • 0: Do not name interfaces using the ifIndex only. • 1: Name interfaces using the ifIndex only.
m_AddIntDisplayLabel		Boolean Integer	Specifies whether to add an interface display label: • 0: Do not add an interface display label. • 1: Add an interface display label.
m_Use_dNCIM		Boolean Integer	By default this field is set to 0. **Important:** Do not change the value of this field. Leave it at the default 0 setting.This field is part of the dNCIM technology preview. For more information on the dNCIM technology preview contact IBM Support.
m_VpnASTagging		Integer	Specifies whether CE facing PE interfaces should be assigned to a private address space: • 0: Do not assign. • 1: Assign. • 2: Automatic. Decides the best approach based on other configuration options.

Table 30. disco.config database table schema (continued)

Column name	Constraints	Data type	Description
m_RefreshDiscovery	Default = 0	Boolean Integer	Specifies whether or not the FullDiscovery stitcher restarts the discovery when called after an initial full discovery has completed. The default value is 0: not to restart the discovery process. Set the value to 1 to restart the discovery process using the RestartDiscoProcess.stch stitcher. Enabling this option can be useful if the discovery is loading custom data into the DiscoContrib.cfg file. The new discovery process reads the file again. It can also help if the discovery process is accumulating memory because the newly started process resets the process to the initial state. **Note:** Even when enabled, the FullDiscovery stitcher only stops and starts the discovery process if there is no discovery in progress at the time it is called.

Related tasks:

"Defining the scope of an MPLS/VPN discovery" on page 113
When configuring the discovery of one or more Virtual Private Networks (VPNs) running across an MPLS core, you can restrict the scope of this discovery to a particular VPN name or VPN Routing and Forwarding (VRF) table name.

Related reference:

"DiscoConfig.cfg configuration file" on page 63
The DiscoConfig.cfg configuration file is used to have the Ping finder automatically check the devices discovered by the File finder, and to enable a context-sensitive discovery.

disco.managedProcesses table

The managedProcesses table is a repository for all the subprocesses managed by DISCO, such as the finders. Provided that CTRL is running, processes that are inserted into this table are started and managed by DISCO.

Table 31. disco.managedProcesses database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The name of the process to be managed.
m_Args		List of text	A list of command-line arguments sent to the executable.
m_Host		Text	The name of the host on which to run the executable.
m_LogFile		Text	The name of the log file to which output is written.

disco.status table

Use the disco.status table to monitor the progress of the **ncp_disco** process during the discovery process.

Attention: The disco.status table is used and updated internally, and you must not make inserts into this table.

Table 32. disco.status database table schema

Column name	Constraints	Data type	Description
m_DiscoveryMode		Integer	The present discovery mode: • 0: Full discovery • 1: Partial discovery
m_Phase		Integer	The current phase within the present discovery cycle. During the data collection stage, the phases are as follows: • 0: The discovery has not yet started. • 1: The main discovery phase in which device data is retrieved. Most discovery agents complete in this phase. • 2 - n: The phases in which the topology data is retrieved for the currently discovered objects. The number of phases required depends on how your discovery is configured. By default, in a layer 2 discovery, phase 2 consists of the retrieval of IP to MAC address translations and phase 3 consists of the retrieval of Ethernet switch topology information. During the data processing stage, the following phase is undertaken. • 3: The phase in which the collected data is processed; the layers are built and the data is sent to MODEL. More detailed information about the discovery phases can be found in “Discovery stages and phases” on page 280.
m_BlackoutState	Externally defined Boolean data type	Boolean Integer	Flag to show if the discovery process is in Blackout mode, that is, whether or not DISCO is accepting new devices from the finders in the current discovery cycle: • 0: False (accepting new devices) • 1: True (not accepting new devices)

Table 32. disco.status database table schema (continued)

Column name	Constraints	Data type	Description
m_CycleCount		Integer	Current® rediscovery cycle, that is, the current number of cycles DISCO has been through without actually building a topology. In rediscovery mode, DISCO only builds a topology at the end of the last cycle (the last cycle is determined by the fact that there is nothing left in `finders.pending` awaiting processing).
m_ProcessingNeeded	Externally defined Boolean data type	Boolean Integer	Flag to indicate whether the current topology needs reprocessing. This flag is checked when DISCO is in rediscovery mode in order to determine whether any newly found devices (that are now in the finders.pending table) necessitate the reprocessing of the entire topology: • 0: The topology does not need reprocessing • 1: The topology needs reprocessing
m_FullDiscovery	Externally defined Boolean data type	Boolean Integer	Flag to indicate that the FullDiscovery.stch stitcher has been called during the discovery. If the stitcher is called, the flag is set to 1 to ensure that the FullDiscovery.stch stitcher is executed when the current discovery finishes (thus starting another full discovery). If the flag is set to any other value, no action is taken.
m_DiscoveryCycle Requested	Externally defined Boolean data type	Boolean Integer	Flag to indicate that a discovery has been requested by the GUI.
m_DiscoveryCycle RequestTime		Integer	The time that the discovery was requested, in Unix time.

disco.agents table

The agents table specifies the discovery agents that DISCO uses for the discovery. Every agent that you want to run must have an insertion into the disco.agents table within the DiscoAgents.cfg configuration file that enables that agent (set m_Valid=1). If m_Valid=0, the agent is not run.

Table 33. disco.agents database table schema

Column name	Constraints	Data type	Description
m_AgentName	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The unique name of the discovery agent.
m_Valid		Integer	A flag determining whether or not the discovery agent is to be used: • (1) Run the discovery agent • (0) Do not run the discovery agent
m_AgentClass		Integer	The category to which the current discovery agent belongs: • (0) Routing agent • (1) Switch agent • (2) Hub agent • (3) ILMI agent • (4) FDDI agent • (5) PNNI agent • (6) Frame Relay agent • (7) CDP agent • (8) NAT agent
m_IsIndirect		Integer	A flag indicating the type of connectivity information returned by the discovery agent: • (0) Direct connectivity; for example, Routing agents • (1) Indirect connectivity information; for example, Switch agents
m_Precedence		Integer	An integer representation of the precedence level of the information returned by the discovery agent; the higher the integer, the higher the weighting given to the information returned. The precedence is only used when there is a conflict when merging device information to produce the workingEntities.finalEntity database table.
m_HostName		Text	The name of the host machine on which to run the agent.
m_DebugLevel		Integer	The level of debugging for the agent.
m_LogFile		Text	The text file to which debugging output is written.

Table 33. disco.agents database table schema (continued)

Column name	Constraints	Data type	Description
`m_NumThreads`		Integer	The number of threads this agent runs. If not specified, the default number is 10; the maximum allowed is 900.
`m_ValidOnPartial`		Integer	Specifies whether the agent is to be used on a partial discovery: • 0: Agent is *not* to be used in a partial discovery. • 1: Agent is to be used in a partial discovery.
`m_MessageLevel`		Text	Specifies the message level (the default is warn). Options include: • debug • info • warn • error • fatal

The disco.agents table also indicates agent precedence, which can be used when merging device information to produce the workingEntities.finalEntity table. Precedence determines which records are used when duplicate or conflicting records are reported by different discovery agents.

The following precedence applies:

- The Details agent has the lowest precedence because it is designed to retrieve only basic device information.
- Routing agents have the next highest precedence. Their connectivity information is at the IP layer only, however, so it is not as accurate as that returned by the switching agents.
- Switching agents have next-highest precedence because they can return information on the media layer (layer 2), which is more accurate than layer 3 information.

Related reference:

"Discovery agent definition files" on page 50
The discovery agent definition files define the operation of the discovery agents.

"DiscoAgents.cfg configuration file" on page 53
The DiscoAgents.cfg configuration file defines which agents run during a discovery.

disco.NATStatus table

The NATStatus table, is used to configure the discovery system to use NAT.

Table 34. disco.NATStatus database table schema

Column name	Constraints	Data type	Description
m_UsingNAT	• UNIQUE • NOT NULL	Boolean integer	This column must be set to indicate whether the discovery uses multiple address spaces. Set this value to: • 1, if the discovery uses multiple address spaces. • 0, if the discovery does not use multiple address spaces.
m_NATStatus	• UNIQUE • NOT NULL	Integer	This column is populated automatically by the discovery process, and can be used to track the process of a NAT discovery. If making inserts into this table, this column must be set to 0. Possible values are: • 0: Uninitialized • 1: Seeded discovery with gateways • 2: Awaiting gateway returns • 3: Processing NAT translations • 4: NAT translations complete

disco.dynamicConfigFiles table

The dynamicConfigFiles table stores the names of configuration files that must be reread each time a full discovery is launched.

Table 35. disco.dynamicConfigFiles database table schema

Column name	Constraints	Data type	Description
m_Name	Primary key Not null	Text	Name of configuration file to be reread,
m_UpdTime		Timestamp	Last update time for this configuration file.

disco.tempData table

The tempData table is used by the discovery profiling stitchers to record the time and memory expended to perform the discovery.

Table 36. disco.tempData database table schema

Column name	Constraints	Data type	Description
m_Phase1TmpTime		Integer	Time taken by phase 1 of the discovery, also known as the Interrogating Devices phase.
m_Phase2TmpTime		Integer	Time taken by phase 2 of the discovery, also known as the Resolving Addresses phase.
m_Phase3TmpTime		Integer	Time taken by phase 3 of the discovery, also known as the Downloading Connections phase.
m_ProcPhaseTmpTime		Integer	Time taken by phase -1, the data processing phase of the discovery, also known as the Correlating Connections phase.
m_Phase1TmpMem		64-character string	Memory used during phase 1 of the discovery.
m_Phase2TmpMem		64-character string	Memory used during phase 2 of the discovery.
m_Phase3TmpMem		64-character string	Memory used during phase 3 of the discovery.
m_ProcPhaseTmpMem		64-character string	Memory used during phase -1 of the discovery.

disco.profilingData table

The profilingData table is used by the discovery profiling stitchers to record data associated with time and memory expended during the discovery.

Table 37. disco.profilingData database table schema

Column name	Constraints	Data type	Description
m_Phase1StartTime		Integer	Time that phase 1 of the discovery started. Phase 1 is also known as the Interrogating Devices phase.
m_Phase2StartTime		Integer	Time that phase 2 of the discovery started. Phase 2 is also known as the Resolving Addresses phase.
m_Phase3StartTime		Integer	Time that phase 3 of the discovery started. Phase 3 is also known as the Downloading Connections phase.
m_ProcPhaseStartTime		Integer	Time that phase -1, the data processing phase of the discovery started. Phase -1 is also known as the Correlating Connections phase.
m_CompletionTime		Integer	Time that phase -1 completed.
m_Phase1StartMem		64-character string	Memory used when phase 1 of the discovery started.
m_Phase2StartMem		64-character string	Memory used when phase 2 of the discovery started.
m_Phase3StartMem		64-character string	Memory used when phase 3 of the discovery started.
m_ProcPhaseStartMem		64-character string	Memory used when phase -1 of the discovery started.
m_CompletionMem		64-character string	Memory used when phase -1 of the discovery completed.

Table 37. disco.profilingData database table schema (continued)

Column name	Constraints	Data type	Description
m_NumFinderInserts		Integer	Total number of finder inserts during the discovery.
m_NumDetailsReturns		Integer	Total number of details table returns during the discovery.
m_NumMainNodes		Integer	Total number of main nodes discovered.
m_NumMainNodesWithAccess		Integer	Total number of main nodes with no SNMP access discovered.
m_NumIPs		Integer	Total number of IP addresses discovered.
m_NumSwitches		Integer	Total number of switches discovered.
m_NumRouters		Integer	Total number of routing devices discovered.
m_NumEntities		Integer	Total number of entities in the scratchTopology database.
m_SoftwareVersion		Text	Software version used.
m_DiscoveryMode		Integer	Type of discovery: • 0: Full discovery • 1: Partial discovery

disco.events table

The events table constrains discovery events generated to a standard format. An event is generated by inserting a record into this table.

Table 38. disco.events database table schema

Column name	Constraints	Data type	Description
m_EventName	Not null	Text	The name of the event.
m_EntityName	Not null	Text	The name of the entity on which the event occurred.
m_EventType	Not null	Integer	This field can take one of the following values: • 1: Problem • 2: Resolution • 13: Informational
m_Severity	Not null	Integer	This field can take one of the following values: • 0: CLEAR • 1: INDETERMINATE • 2: WARNING • 3: MINOR • 4: MAJOR • 5: CRITICAL It is possible to define more values.
m_Description	Not null	Text	Description of the discovery event
m_ExtraInfo	Externally defined vblist data type		Specifies a list of additional information.

Related concepts:

"Process flow for creating discovery events" on page 164
Discovery events are created during the discovery process showing the progress of agents, stitchers, and finders. These events are sent to and stored in Tivoli Netcool/OMNIbus and can be viewed using the Web GUI.

disco.ipCustomTags table

The ipCustomTags table stores custom tags, which can be associated with unique discovered entities during the discovery and used to perform custom visualization and polling tasks.

Table 39. disco.ipCustomTags database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	Not null	Text	IP address to which the name-value pair tags in the m_CustomTags is to be associated to.
m_StitcherTagName	Not null	Text	Name of a tag to be evaluated using the GetTagStitcher.stch stitcher.
m_CustomTags	Not null	Object type vblist	List of name-value pair tags.

Related tasks:

"Adding tags to entities using custom tag tables" on page 173
You can add name-value pair tags to entities by creating inserts containing the name-value pair data into the disco.ipCustomTags table or into the disco.filterCustomTags table.

disco.filterCustomTags table

The filterCustomTags table stores custom tags, which can be associated with a filtered set of discovered entities during the discovery and used to perform custom visualization and polling tasks.

Table 40. disco.filterCustomTags database table schema

Column name	Constraints	Data type	Description
m_Filter	Not null	Text	Filter definition that extracts a set of IP address to which the name-value pair tags in the m_CustomTags is to be associated to. You can create a filter based on any attributes associated with discovered entities. For example, you could apply the following filters: • Filter based on entity IP address: "m_UniqueAddress LIKE '172.20.3'" • Filter based on entity name: "m_Name LIKE 'lon'" • Filter based on VLAN identifier of a VLAN entity: "m_LocalNbr->m_VlanID = 102"
m_StitcherTagName	Not null	Text	Name of a tag to be evaluated using the GetTagStitcher.stch stitcher.
m_CustomTags	Not null	Object type vblist	List of name-value pair tags.

Related tasks:

"Adding tags to entities using custom tag tables" on page 173
You can add name-value pair tags to entities by creating inserts containing the name-value pair data into the disco.ipCustomTags table or into the disco.filterCustomTags table.

Example configuration of the disco.config table

This example uses OQL commands to insert configuration values into the disco.config table.

- The maximum period between device discovery is 300 seconds. This condition and the next condition must be satisfied in order to proceed to the next phase of the discovery cycle.
- The maximum allowable ratio of pending to processing devices is 20 percent. If this threshold is breached, a full discovery is instigated.
- The cycle limit is 5, which means that a maximum of five discovery cycles are necessary to complete the discovery process. If there are more than 5 discovery cycles, a full rediscovery is initiated.
- The agent restart flag is 1, which means that DISCO is mandated to restart any discovery agent that fails in its operation.
- The finder restart flag is 1, which means that DISCO is mandated to restart any finder that fails in its operation.
- Scans for updates to the agents and stitchers have been disabled. This is usually the case when you do not wish to alter the discovery data flow.
- Do not write a cache of the Discovery engine, ncp_disco, tables to disk.

```
insert into disco.config
(
          m_NothngFindPeriod,
          m_PendingPerCent,
          m_CycleLimit,
          m_RestartAgents,
          m_RestartFinders,
          m_DirScanIntvl
          m_WriteTablesToCache
)
values
(
          300,
          20,
          5,
          1,
          1,
          0,
          0
);
```

Example configuration of the disco.managedProcesses table

This example uses OQL commands to insert configuration values into the disco.managedProcesses table. If the CTRL program is running, you can configure, launch, and manage the File finder and Ping finder subprocesses.

```
insert into disco.managedProcesses
(
          m_Name, m_Args, m_Host
)
values
(
          "ncp_df_file", [ ], "othello"
);
```

```
insert into disco.managedProcesses
(
            m_Name, m_Args, m_Host
)
values
(
            "ncp_df_ping", [ ], "othello"
);
```

Example configuration of the disco.agents table

This example uses OQL commands to insert configuration values into the disco.agents table.

- The ArpCache discovery agent is enabled to run during the discovery (m_Valid=1), belongs to the routing class (m_AgentClass=0), returns direct connectivity information (m_IsIndirect=0) and has a precedence level of 2.
- The AtmForumPnni discovery agent is disabled for this discovery (m_Valid=0), belongs to the PNNI class (m_AgentClass=5), returns direct connectivity information (m_IsIndirect=0) and has a precedence level of 5.
- The BayEthernetHub discovery agent is disabled for this discovery (m_Valid=0), belongs to the hub class (m_AgentClass=2), returns indirect connectivity information (m_IsIndirect=1) and has a precedence level of 3.

```
insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
)
values
(
        'ArpCache', 1, 0, 0, 2
);

insert into disco.agents
(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
)
values
(
        'AtmForumPnni', 0, 5, 0, 5
);

insert into disco.agents

(
        m_AgentName, m_Valid, m_AgentClass, m_IsIndirect, m_Precedence
)
values
(
        'BayEthernetHub', 0, 2, 1, 3
);
```

Discovery scope database

The scope database limits the extent or reach of a discovery. Using the scope database, you can configure a range of protocols and attributes that define zones that are to be included or excluded from the discovery process.

The range of IP addresses and devices that can potentially be considered by the discovery process is unlimited, so unless you restrict the scope of the discovery, ncp_disco would eventually attempt to discover the entire Internet.

For example, you can specify that sensitive devices not be discovered and consequently not be instantiated. A sensitive device is one that you do not want to poll. This might be because there is a security risk involved in polling the device, or that polling might overload device.

Related reference:

"DiscoScope.cfg configuration file" on page 64
The DiscoScope.cfg configuration file can be used to configure the scope of a discovery.

disco.scope database schema

The scope database is defined in $NCHOME/etc/precision/DiscoSchema.cfg and $NCHOME/etc/precision/DiscoScope.cfg. Its fully qualified database table names are: scope.zones; scope.detectionFilter; scope.instantiateFilter; scope.special.

scope.detectionFilter table

If you specify a filter in the detectionFilter table, only devices matching it are discovered. Because the m_Protocol column must be unique, there must be only one insert into this table for any given protocol. Multiple filters must be defined within a single insert.

Table 41. scope.detectionFilter database table schema

Column name	Constraints	Data type	Description
m_Protocol	• PRIMARY KEY • UNIQUE • NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of network protocol used by the presently defined zone. Currently only IP is supported: • 0: Undefined • 1: IP
m_Filter		Text	A textual representation of an attribute filter against the columns of the Details.returns table; for example, m_UniqueAddress or m_ObjectId.

Although you can configure the filter condition to test any of the columns in the Details.returns table, you might need to use the IP address as the basis for the filter if you need to restrict the detection of a particular device. If the device does not grant SNMP access to the Details agent, the Details agent might not be able to retrieve MIB variables such as the Object ID. However, you are guaranteed the return of at least the IP address when the device is detected.

inferMPLSPEs table

Use the inferMPLSPEs table when enabling inference of inaccessible provider-edge (PE) devices by using the BGP data on the customer-edge (CE) devices. This table enables you to optionally specify which zones to process to determine which of the inferred PE devices are valid devices.

To specify which zones to process to determine which of the inferred PE devices are valid devices populate the scope.inferMPLSPEs table, using standard format scope entries, as in the scope.zones table. Use this option when you have inaccessible devices that are connected by means of BGP but which are not actually PE devices.

If the following conditions are true, then the system creates a "third-party" network object to model this inaccessible provider network.

- A router is within this scope
- The router has BGP peers outside the discovered network
- m_InferMPLSPEsUsingBGP is on. This can also be defined using the **Advanced** tab on the Discovery Configuration GUI.

Table 42. scope.inferMPLSPEs database table schema

Column name	Constraints	Data type	Description
m_Protocol	• PRIMARY KEY • NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently defined zone. Currently only IP is supported: • 0: Undefined • 1: IP
m_Action	• NOT NULL • Externally defined filtAction data type	Integer	Action to perform for current zone: • 0: Undefined • 1: Include • 2: Exclude
m_Zones	NOT NULL	List of type zone	A list of varbinds (name=value) that define the present discovery zone.

Only process interfaces in the 199.220.* network

The following example shows how to instruct the system to only process interfaces in the 199.220.* network.

```
insert into scope.inferMPLSPEs
(
     Protocol,
     m_Action,
     m_Zones
)
values
(
     1,
     1,
     [ { m_Subnet = "199.220.*" } ]
//);
```

scope.instantiateFilter table

When you specify a filter in the instantiateFilter table, only devices that pass the criteria are instantiated, that is, sent to MODEL. If no filter is specified, all discovered devices are instantiated.

Note that because the m_Protocol column must be unique, there must be only one insert into this table for any given protocol. Multiple filters must be defined within a single insert.

Table 43. scope.instantiateFilter database table schema

Column name	Constraints	Data type	Description
m_Protocol	• PRIMARY KEY • UNIQUE • NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently-defined zone. Currently only IP is supported: • 0: Undefined • 1: IP
m_Filter		Text	A textual representation of an attribute filter against the columns of the scratchTopology.entityByName table; for example, EntityOID or Address.

mplsTe table

The mplsTe table defines the scope of MPLS Traffic Engineered (TE) tunnel discovery, and defines what information is retrieved.

The following table shows the schema of the scope.mplsTe table.

Table 44. scope.zones database table schema

Column name	Constraints	Data type	Description
m_Protocol	• NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently defined zone. The following values are possible: • 0: Undefined • 1: Internet Protocol (IP) • 2: Network Address Translation (NAT) • 3: IPv6
m_Zones	NOT NULL	List of type zone	Defines the scope in which the tunnel heads will be discovered
m_AddressSpace		Text	Optional address space

Table 44. scope.zones database table schema (continued)

Column name	Constraints	Data type	Description
m_Mode		Integer	The TE tunnel discovery mode defines what information is retrieved. Possible values are: • 0: Unknown (not set) • 1: Tunnel Head/Tail with Transit Hop List • 2: Tunnel Head/Tail (No Hop List) • 3: Tunnel Head, Tails, and Transit devices
m_TunnelFilter		Integer	The TE tunnel filter. Possible values are: • 0: Unknown (not set) • 1: Include tunnels with this head • 2: Exclude tunnels with this head

Related tasks:

"Configuring the StandardMPLSTE agent" on page 112
Configure which tunnels to discover, and what details to retrieve.

scope.multicastGroup table

The scope.multicastGroup table defines which multicast groups to discover and which details to retrieve from these groups.

The following table shows the schema of the scope.multicastGroup table.

Table 45. scope.multicastGroup database table schema

Column name	Constraints	Data type	Description
m_AddressSpace		Text	Optional address space
m_GroupName		Text	Descriptive name for a group
m_Groups	Not null	list type zone	Zones defines the multicast subnets to which the scope options apply
m_IGMPMode		Integer	IGMP Group discovery mode • 0 - unknown (use default) • 1 - Include group • 2 - Exclude group
m_IPMRouteMode		Integer	IP Multicast Route Group discovery mode: • 0 - unknown (use default) • 1 - Include group • 2 - Exclude group

Table 45. scope.multicastGroup database table schema (continued)

Column name	Constraints	Data type	Description
m_PimMode		Integer	The PIM multicast discovery mode defines what information is retrieved. Possible values are: • 0: Unknown (use default) • 1: Retrieve PIM group data • 2: Do not retrieve PIM group data. Groups with this option applied will not be represented in the PIM Service/End Point data.
m_Protocol	• NOT NULL • Externally defined netProtocol data type (Currently IPv4 [1] only)	Integer	An integer representation of the network protocol used by the presently defined group. The following values are possible: • 0: Undefined • 1: IP • 2: NAT • 3: IPv6

Related tasks:

"Configuring a multicast discovery" on page 34
Configure a multicast discovery by enabling the required agents and scoping the discovery.

scope.multicastSource table

The scope.multicastSource table defines which IPM routes to discover. This is particularly useful if you have multiple IPM route sources, since you can scope multicast discovery by IPM route source to focus on the sources of interest.

The following table shows the schema of the scope.multicastSource table.

Table 46. scope.multicastSource database table schema

Column name	Constraints	Data type	Description
m_Protocol	• NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently defined group. The following values are possible: • 0: Undefined • 1: IP • 2: NAT • 3: IPv6
m_Source	NOT NULL	list type zone	The multicast source to be included or excluded

Table 46. scope.multicastSource database table schema (continued)

Column name	Constraints	Data type	Description
m_IPMRouteMode		Integer	An integer representation of the network protocol used by the presently defined group. The following values are possible: • IP Multicast Route Source discovery mode • 0 - unknown (use default) • 1 - Include Source • 2 - Exclude Source
m_Groups		list type zone	The multicast group subnets to which the source scope option applies

Related tasks:

"Configuring a multicast discovery" on page 34
Configure a multicast discovery by enabling the required agents and scoping the discovery.

scope.special table

The special table defines management IP addresses. A management address is an IP address on a device whose only role is to manage the device. Management addresses do not handle network traffic.

Table 47. scope.special database table schema

Column name	Constraints	Data type	Description
m_Zones	NOT NULL	List of type zone	A list of varbinds (name=value) that define the present discovery zone. This takes the form of a list of subnet IP addresses and subnet.
m_AddressSpace		Text	Optional address space identifier for a particular scope entry.
m_Protocol		Integer	Protocol of the network. Takes one of the following values: • 0: Undefined • 1: IP • 2: NAT • 3: IPv6
m_OutOfBand		Int Type Boolean	Indicates whether the management area is out of band. Takes one of the following values: • 0: in band • 1: out of band
m_IsManagement		Int Type Boolean	Indicates whether the address is a management address.

Table 47. scope.special database table schema (continued)

Column name	Constraints	Data type	Description
m_IsValidVirtual		Int Type Boolean	Indicates whether the address is a valid virtual IP.

scope.zones table

Use the zones table to define areas of the network to be either included or excluded from the discovery process. A zone is typically defined as a list of varbinds. Varbinds are `name = value` pairs.

You can define multiple zones, and you can combine inclusion and exclusion zones. However, if you define a combination of inclusion and exclusion zones, the exclusion zones must be within the scope of the inclusion zones.

Table 48. scope.zones database table schema

Column name	Constraints	Data type	Description
m_Protocol	• PRIMARY KEY • NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently defined zone. Currently only IP is supported: • 0: Undefined • 1: IP
m_Action	• NOT NULL • Externally defined filtAction data type	Integer	Action to perform for current zone: • 0: Undefined • 1: Include • 2: Exclude
m_Zones		List of type zone	A list of varbinds (name=value) that define the present discovery zone.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

Example scope database configuration

The example OQL inserts into the scope database tables in this topic would be appended to the DiscoScope.cfg configuration file to configure DISCO when it is launched.

Tip: In the detectionFilter and the instantiateFilter tables of the scope database, the m_Protocol column is UNIQUE. Therefore, there must be no more than one insert into either table per protocol.

Configuration of the scope.zones table

Use this information to understand how to configure the scope.zones table.

Creating two inclusion zones

This example configuration of the scope.zones table creates two inclusion zones for the current discovery. Both zones are defined using a single insert.

```
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
(
        1,
        1,
        [
            {
                m_Subnet="172.16.1.0"
                m_NetMask=24
            },
            {
                m_Subnet="172.16.2.*"
            }
        ]
);
```

The previous OQL insert specifies the following conditions:

- The network uses the Internet Protocol (m_Protocol=1).
- Any devices that fall into the present zone are to be included in the discovery (m_Action=1).
- The discovery includes:
  - Any device that falls within the 172.16.1.0 subnet (with a subnet mask of 24, that is, 24 bits turned on and 8 bits turned off, which implies a netmask of 255.255.255.0).
  - Any device with an IP address starting with 172.16.2, that is, in the 172.16.2.0 subnet with a mask of 255.255.255.0.

Creating a zone within a zone

Zones can be specified within zones: within a given inclusion zone, you can specify devices or subnets that are not to be detected. These devices are not pinged by the Ping finder or interrogated by the discovery agents. For example, you can define an include scope zone consisting of the Class B subnet 172.20.0.0/16, and completely contained within that zone you can specify an exclude scope zone consisting of the subnet 172.20.32.0/19. Finally, completely contained within the exclude scope zone you could specify an include scope zone 172.20.33.0/24.

```
// Include all IP addresses in this range
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
(
        1,
        1,
       [{m_Subnet = '172.20.0.0', m_NetMask = 16 }]
);

// Apart from the IP addresses in this range
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
(
        1,
        2,
        [{m_Subnet = '172.20.32.0' , m_NetMask = 19 }]
);
// Except for these IP addresses which we do want to include
insert into scope.zones
(
        m_Protocol,
        m_Action,
        m_Zones
)
values
(
        1,
        1,
        [{m_Subnet = '172.20.33.0' , m_NetMask = 24 }]
);
```

The previous OQL insert specifies three scope zones:

- All zones specify that the network uses the Internet Protocol (m_Protocol=1).
- Include and exclude zones are defined as follows:
  - Any devices that fall into the first zone, 172.20.0.0/16, are to be included in the discovery (m_Action=1).
  - Any devices that fall into the second zone, 172.20.32.0/19, which is completely contained within the first zone, are to be excluded from the discovery (m_Action=2).
  - Any devices that fall into the third zone, 172.20.33.0/24, which is completely contained within the second zone, are to be included in the discovery (m_Action=1).

Preventing the detection of devices with a filter

This example insert defines a detection filter. Since there must only be one insert into the scope.detectionFilter table, multiple conditions for IP must be defined using a single insert. The conditions of the filter can be combined using the Boolean OQL keywords AND and OR.

```
insert into scope.detectionFilter
(
        m_Protocol, m_Filter
)
values
(
        1,
        "(
                ( m_UniqueAddress <> '10.10.63.234' )
                AND
                ( m_ObjectId not like '1\.3\.6\.1\.4\.1\..*' )
        )"
);
```

The above example filter ensures that only the following are further interrogated by the discovery:

- Devices that do not have the IP address `10.10.63.234`.
- Devices that do not have the Object ID `1.3.6.1.4.1.*`.

In the above example, the backslash (\) is used in conjunction with the `not like` comparison to escape the `.` character, which would otherwise be treated as a wildcard.

Restricting instantiation based on Object ID

This example insert defines an instantiation filter. This example prevents the instantiation of devices that match a given Object ID (OID).

The filter (m_Filter) uses column values from the scratchTopology.entityByName table.

Note: To ensure that alerts are not raised for *interfaces* that are excluded by the instantiation filter, you must set the `RaiseAlertsForUnknownInterfaces` variable. To this, perform the following steps:

1. Edit the `$NCHOME/etc/precision/NcPollerSchema.cfg` configuration file.
2. Add the following line to the file:

   ```
   update config.properties set RaiseAlertsForUnknownInterfaces = 1;
   ```

Restricting instantiation based on Object ID

The OQL clause, `not like`, indicates that only devices that pass the filter (that is, devices for which the OID is not like 1.3.6.1.4.1.*) are instantiated.

```
insert into scope.instantiateFilter
(
        m_Protocol,
        m_Filter
)
values
(
        1,         // The backslash is used here to escape the .
        "(         // which would otherwise be treated
                   // as a wildcard.
                ( EntityOID not like '1\.3\.6\.1\.4\.1\..*' )
        )"
);
```

Access databases

There are several databases that control access to network devices: snmpStack database and telnetStack database.

snmpStack database

The snmpStack database defines the operation of the SNMP helper.

Description

The snmpStack database in defined in the `SnmpStackSchema.cfg` file.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

snmpStack.accessParameters database table

The snmpStack.accessParameters database table configures the way that the SNMP helper handles the retrieval of large non-scalar variables for particular devices or subnets.

Description

Any values inserted into this table override the values for `m_GetNextBoundary` and `m_GetNextSlowDown` that have been specified in the `snmpHelper.configuration` table.

Schema

The snmpStack.accessParameters database table schema is described in the following table:

Table 49. snmpStack.accessParameters database table schema

Column name	Constraints	Data type	Description
`m_NetAddress`	NOT NULL	Text	The IP address on which to override the boundary and slowdown values.
`m_NetMask`		Text	The netmask. If no netmask is specified, `m_NetAddress` is taken to be a single IP address. If a netmask is specified, `m_NetAddress` is taken to be a subnet address.
`m_GetNextSlowDown`	NOT NULL	Integer	The delay (in milliseconds) to introduce between each SNMP GetNext request when the number of separate GetNext requests issued while retrieving a particular non-scalar SNMP variable exceeds `m_GetNextBoundary`.
`m_GetNextBoundary`	NOT NULL	Integer	When retrieving a particular non-scalar SNMP variable from a device, this is the minimum number of GetNext requests to be issued before the delay specified by `m_GetNextSlowDown` is introduced.

Table 49. snmpStack.accessParameters database table schema (continued)

Column name	Constraints	Data type	Description
m_GeneralSlowDown	NOT NULL	Integer	The general amount by which to delay a request (in milliseconds). A general slow down must only be used where absolutely necessary as it can significantly increase the overall discovery time.
m_useGetBulk	NOT NULL	Boolean Integer	Indicates whether the SNMP Helper must use GetBulk when processing devices using SNMP v2 or SNMP v3. This field can take the following values: • 0: Do not use GetBulk • 1: Use GetBulk

snmpStack.configuration database table

The snmpStack.configuration table controls the general operation of the SNMP helper.

Schema

The snmpStack.configuration database table schema is described in the following table:

Table 50. snmpStack.configuration database table schema

Column name	Constraints	Data type	Description
m_AutoVersion	Externally defined Boolean data type	Boolean Integer	Flag controlling automatic SNMP versioning: • 1: Use automatic SNMP versioning. The SNMP helper initially attempts to use SNMP V3 for device access; if unsuccessful, it uses SNMP V2, then SNMP V1. • 0: Do not use automatic versioning. The SNMP helper ignores the entries in the versions table.
m_AllowOQL	Externally defined Boolean data type	Boolean Integer	Flag controlling OQL access to the SnmpHelper database: • 1: Allowed OQL access to the cached community strings for the devices discovered. • 2: Do not allow OQL access.
m_ExpireAfter		Long	The time, in seconds, after which to expire the cached community string for device if it has not been used. The default value of zero does not expire cache community strings.

snmpStack.conversion database table

The snmpStack.conversion database table configures the SNMP Helper to replace characters that are not allowed in the locale of the NCIM database with the question mark character: '?'.

Description

The SNMP Helper substitutes characters on only those objects that are configured in the snmpStack.multibyteObjects table.

Inserts into this database table are configured in the `SnmpStackSchema.cfg` file.

Schema

The snmpStack.conversion database table schema is described in the following table:

Table 51. snmpStack.conversion database table schema

Column name	Constraints	Data type	Description
`m_SubstituteInvalidUTF8`	NOT NULL	Integer	If set to 1, the SNMP Helper replaces characters that are not allowed in the locale of the NCIM database with the question mark character: '?'. If set to 0, no action is taken on invalid characters.

snmpStack.multibyteObjects table

The snmpStack.multibyteObjects table defines MIB objects that are checked to see if they are multibyte strings.

Description

Sending a raw ASCII string back to the helper server can cause problems if the string contains characters with special meaning in ASCII. If the MIB objects contain multibyte strings, the SNMP helper encodes them.

Schema

The snmpStack.multibyteObjects database table schema is described in the following table:

Table 52. snmpStack.multibyteObjects database table schema

Column name	Constraints	Data type	Description
m_ObjectName	NOT NULL	Text	The MIB object name to be checked.

snmpStack.verSecurityTable database table

The snmpStack.verSecurityTable maps an IP or subnet address with an SNMP version (1, 2, or 3).

Description

The security parameters must be configured, as specified by the SNMP version, in order to gain SNMP access to network device. An example of this is the use of community strings for SNMP versions 1 and 2, as well as the specification of the different security levels offered by SNMP V3.

Schema

The snmpStack.verSecurityTable database table schema is described in the following table:

Table 53. snmpStack.verSecurityTable database table schema

Column name	Constraints	Data type	Description
`m_IpOrSubNetVer`		Text	The IP or subnet address to which the device access configuration specified by this record is applicable. The interpretation of this field as an IP or a subnet address is dependent on the value specified in the `m_NetMaskBitsVer` field.
`m_NetMaskBitsVer`		Integer	The subnet mask for the address specified by the `m_IpOrSubNetVer` field. If this field is set to 32 then `m_IpOrSubNetVer` is taken as a single IP address.
`m_SNMPVersion`		Integer	The SNMP version that this configuration applies to. • 0: SNMP V1 • 1: SNMP V2 • 2: SNMP V3
`m_Password`		Text	The password to try for this IP or subnet address; for example, community string.
`m_Type`		Integer	An integer classification of the password type; for example: (2) SNMP Get password.
`m_SNMPVer3Level`		Integer	Integer representation of the SNMP V3 security level.
`m_SNMPVer3Details`		Object of type V3SecInfo	An object representation of the authpassword and/or privpassword details for SNMP V3.
`m_SecurityName`		Text	The SNMP V3 security password.

Table 53. snmpStack.verSecurityTable database table schema (continued)

Column name	Constraints	Data type	Description
m_SnmpPort		Integer	The SNMP port on the target device, or target devices if the device access configuration specified by this record is applicable to a subnet. If no value is specified for m_SnmpPort, then the value defaults to the standard SNMP 161 port.

telnetStack database

The telnetStack database defines the Telnet access parameters for devices.

Description

The telnetStack database is defined in the TelnetStackSchema.cfg file. It has the following tables:

- telnetStack.configuration
- telnetStack.passwords

Related reference:

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

telnetStack.passwords database table

The telnetStack.passwords database table defines the Telnet access parameters for devices.

Schema

The telnetStack.passwords database table schema is described in the following table:

Table 54. telnetStack.passwords database table schema

Column name	Constraints	Data type	Description
m_IpOrSubNet		Text	IP or subnet address depending on value of m_NetMaskBits.
m_NetMaskBits		Integer	The subnet mask. If set to 32, m_IpOrSubNet is taken to be a single IP address.
m_Password	NOT NULL	Text	The password to try for this subnet or IP address. Default = "\n" (carriage return).
m_Username		Text	The username to try for this subnet or IP address. Default = "".
m_PwdPrompt		Text	Password prompt to expect from remote device. Default = ".*assword:.*".

Table 54. telnetStack.passwords database table schema (continued)

Column name	Constraints	Data type	Description
m_LogPrompt		Text	Login prompt to expect from remote device. Default = ".*ogin:.*".
m_ConPrompt		Text	Console prompt to expect from remote device. Default = "^.*[a-zA-Z0-9].*[$ %>#]$".
m_SSHSupport		Boolean Integer	Flag to indicate whether or not to use SSH support for this device: • 1: Use SSH support for this device. • 0: Do not use SSH support for this device. If no value is specified for m_SSHSupport, then the value defaults to 0, that is, no SSH support.

Process management databases

On startup, the discovery engine, ncp_disco, populates the agent and stitcher databases with information extracted from the discovery agent and discovery stitcher files. While operating, ncp_disco scans for alterations to the agent and stitcher files and updates the agent and stitcher databases if necessary. The frequency of scans is set in the disco database.

The agents and stitchers databases contain definition and configuration information for the agents and stitchers, such as a list of the types of devices that are sent to any given agent. The information in these databases is extracted by the discovery engine from the following directories:

- /precision/disco/agents
- /precision/disco/stitchers

The stitchers databases also contain information about when any given stitcher is triggered; for example, "start stitcher X upon completion of agent Y," or "start stitcher X upon the insertion of an entry into database table Z." It is therefore possible to start stitchers on demand by inserting their name into the appropriate actions table using OQL. The necessary agents are started automatically when a device is inserted into the despatch table of the agent.

Configuring the data flow: starting stitchers on-demand

The information extracted by DISCO contains the full definitions of the agents and stitchers, which includes the trigger conditions. By modifying the trigger conditions, you can modify the data flow of the discovery process.

You can start the discovery cycle from any point within the configured data flow by placing a stitcher into the actions table of the stitchers database.

agents database schema

The agents database is defined in `$NCHOME/etc/precision/DiscoSchema.cfg`. Its fully qualified database table names are: agents.definitions; agents.victims; agents.status

agents.definitions table

The agents.definitions table contains scheduling information for every discovery agent, extracted from the information in the discovery agent file.

Table 55. agents.definitions database table schema

Column name	Constraints	Data type	Description
`m_Name`	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the agent.
`m_Type`	Externally defined agentType data type	Integer	Agent Type: • 0: Undefined • 1: Precompiled • 2: Text defined • 3: Combination
`m_Text`	NOT NULL	Text	Textual description of agent rules.
`m_ExecuteOn`		Text	The host on which to execute the agent.
`m_Phase`	Default = 1	Integer	The discovery phase by the end of which the agent is expected to complete.
`m_UpdTime`		Long integer	The time of the last modification, which determines whether the agent has changed since its definition was stored.

agents.victims table

The agents.victims table contains an extraction of the criteria that determine which devices get sent to the agent.

Table 56. agents.victims database table schema

Column name	Constraints	Data type	Description
`m_Name`	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the agent.
`m_Filter`		Text	The filter condition that determines which devices are sent to the agent.

agents.status table

The agents.status table contains information about the present status of the agent.

Table 57. agents.status database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the agent.
m_State	Externally defined agentState data type. Default = 0	Integer	The current state of the agent: • 0: Undefined • 1: Not running • 2: Start up • 3: Running • 4: Finished • 5: Died
m_NumConnects	Default = 0	Integer	The number of times that DISCO has connected to the agent.

Stitchers database schema

The stitchers database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table names are: stitchers.definitions; stitchers.triggers; stitchers.status; stitchers.actions.

stitchers.definitions table

The stitchers.definitions table contains the scheduling information for every discovery stitcher.

Table 58. stitchers.definitions database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the stitcher.
m_Type	Externally defined stitcherType data type	Integer	Stitcher type: • 0: Undefined • 1: Precompiled • 2: Text defined
m_Text		Text	Textual description of stitcher rules.
m_Phase	Default = 0	Integer	The discovery phase by the end of which the stitcher is expected to complete.
m_UpdTime		Long integer	The time of the last modification to the stitcher.

stitchers.triggers table

The stitchers.triggers table contains an extraction of the criteria that determine the trigger for the stitcher.

Table 59. stitchers.triggers database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the stitcher.
m_Type		Integer	The type of stitcher trigger: • 0: Undefined • 1: On the completion of some other activity; for example, another stitcher or a discovery phase • 2: On a table insert • 3: On demand • 4: On a timer
m_Trigger	Externally defined ruleTrigger data type	Object	Description of the stitcher trigger.

stitchers.status table

The stitchers.status table contains the information about the present status of the stitcher.

Table 60. stitchers.status database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Name of the stitcher.
m_State	Externally defined stchrState data type Default = 0	Integer	The current state of the stitcher: • 0: Undefined • 1: Start up • 2: Running • 3: Finished • 4: Not maintained (the stitcher is not having its state maintained)

stitchers.actions table

If a stitcher is inserted into the stitchers.actions table, DISCO runs the stitcher. Once the stitcher has completed, its entry is deleted from the stitchers.actions table. Any stitchers triggered to execute from the stitcher that has been inserted, or upon completion of the stitcher, are also executed.

You can also configure other actions to take place on completion of the stitcher, so that the discovery cycle completes from that point onwards.

Table 61. stitchers.actions database table schema

Column name	Constraints	Data type	Description
`m_Name`	• PRIMARY KEY • NOT NULL	Text	Name of the stitcher.

Related concepts:

"Configurable discovery data flow" on page 294
The discovery process data flow is user-configurable. Stitchers control the movement of data between databases, and you can customize the discovery process by changing the way in which the stitchers are triggered and operate.

Subprocess databases

The finders, Details, and agent databases are used during the discovery by the discovery engine subprocesses to store information retrieved from the network. The databases are defined within the configuration file, DiscoSchema.cfg.

The subprocess databases include:

- The finders database, which is used by the finders to store information about device existence.
- The Details database, which is used by the Details agent to store basic device information.
- The discovery agent databases, which are created using a template.

The finders, Details and AssocAddress agents must always be run, so their databases are defined in the DiscoSchema.cfg configuration file. The databases for the rest of the discovery agents are created based on a template that is defined in the DiscoSchema.cfg configuration file.

finders database schema

The finders database is defined in $NCHOME/etc/precision/DiscoSchema.cfg.

The fully qualified database table names of the finders database are:

- finders.despatch
- finders.returns
- finders.pending
- finders.processing
- finders.rediscovery

The finders database is the central monitoring and management point for finders operating during discovery. The finders discover the existence of devices and report these devices back to the finders database, but do not discover connections.

Network entities reported by the finders are usually sent to the Details agent for retrieval of basic device information, although the discovery data flow is fully configurable.

Related concepts:

"Discovery cycles" on page 285
A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

finders.despatch table

The finders.despatch table contains a record of all the requests sent to the finders and the current status of the requests.

Table 62. finders.despatch database table schema

Column name	Constraints	Data type	Description
`m_Finder`	• PRIMARY KEY • NOT NULL	Text	The name of the finder responsible for the request.
`m_FindRequest`	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The OQL request sent to the finder named above.
`m_Request Status`		Integer	The current status of the request sent to the finder.

finders.returns table

When a finder finds a device, it returns the information to the finders.returns table, provided that the discovery is still in the device discovery phase, that is, data collection phase one. If the discovery is in the blackout state, the finders return the information to the pending table.

The returns table serves as a transfer point, notifying the system that a device exists. By default, a stitcher sends the device information to the Details agent to discover basic device information.

Table 63. finders.returns database table schema

Column name	Constraints	Data type	Description
`m_UniqueAddress`	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The IP address of the discovered network entity.
`m_Name`		Text	The unique name of the network entity.
`m_Creator`		Text	The finder that created this record.
`m_Protocol`		Integer	The protocol of the discovered device: • 1: IP • 2: IP-NAT

finders.pending table

The pending table accepts device information when the returns table has been locked out by DISCO. The returns table has to be locked during data processing because even though the data collection stage has completed, it does not necessarily mean that all the devices on the network have been discovered.

Network entities that have been sent to the pending table are processed after the current discovery cycle has been completed.

Table 64. finders.pending database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The IP address of the discovered network entity.
m_Name		Text	The unique name of the network entity.
m_Creator		Text	The finder that created this record in the table.
m_Protocol		Integer	The protocol of the discovered device: • 1: IP • 2: IP-NAT
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

finders.processing table

The processing table contains a record of all the discovered entities that are currently being processed by DISCO. Any device that has been reported to the returns table and is awaiting the next action to take place has an entry in the processing table.

Table 65. finders.processing database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	• PRIMARY KEY • UNIQUE • NOT NULL	Text	The IP address of the discovered network entity.
m_Name		Text	The unique name of the network entity.
m_Creator		Text	The finder that created this record in the table.

Table 65. finders.processing database table schema (continued)

Column name	Constraints	Data type	Description
`m_Protocol`		Integer	The protocol of the discovered device: (1) IP (2) IP-NAT
`m_AddressSpace`		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

finders.rediscovery table

The rediscovery table can hold nodes and subnets that you want to rediscover. Any device inserted into this table is sent to the Ping finder for processing.

Table 66. finders.rediscovery database table schema

Column name	Constraints	Data type	Description
`m_Address`	• PRIMARY KEY • NOT NULL	Text	The address to ping.
`m_RequestType`		Int	The type of IP address: • 1: Individual • 2: Subnet
`m_NetMask`		Text	The net mask if the address refers to a subnet.
`m_Protocol`	NOT NULL	Int	The protocol of this IP address: • 1: IPv4 • 3: IPv6

Details database schema

The Details database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table names are: `Details.despatch`; `Details.returns`.

The Details agent retrieves basic information about devices discovered by the finders when information from the finders is placed in the despatch table. The Details agent retrieves the appropriate device information and places the results in the returns table.

A stitcher takes the information from the Details.returns table and sends it to the Associated Address agent and ultimately the appropriate discovery agent.

details.despatch table

The despatch table contains basic information about devices that have been detected by the finders. When data is placed in this table, the Details agent automatically interrogates the network for more detailed device information.

Table 67. Details.despatch database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	• PRIMARY KEY • NOT NULL	Text	Unique IP address of the network entity.
m_Name		Text	Unique name of an entity on the network.
m_Protocol		Integer	The protocol of the discovered device: • 1: IP • 2: IP-NAT
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

details.returns table

The returns table holds detailed device information retrieved by the Details agent. Information inserted into this table is automatically processed by the stitchers so that the device connectivity can be discovered by the appropriate discovery agent.

Table 68. Details.returns database table schema

Column name	Constraints	Data type	Description
m_Name		Text	Unique name of an entity on the network.
m_UniqueAddress	NOT NULL	Text	Layer 3 address.
m_Protocol		Integer	The protocol of the discovered device: • 1: IP • 2: IP-NAT
m_ObjectId		Text	Textual representation of the device class (an ASN.1 address).
m_Description		Text	Value of sysDescr MIB variable of the entity.
m_HaveAccess	Externally defined Boolean data type	Integer	Flag indicating whether there is SNMP access to the device: • 1: Have access • 0: No access
m_UpdAgent		Text	The agent that updated this device.

Table 68. Details.returns database table schema (continued)

Column name	Constraints	Data type	Description
m_LastRecord	Externally defined Boolean data type	Boolean integer	A flag indicating whether this is the last record for this entity (that is, whether the entity has been completely processed): • 1: True • 0: False
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
m_ExtraInfo	Externally defined vblist data type	Object	Any extra information.

Finders databases

Finders determine device existence. Each of the finders uses a different method to discover network devices. You can enable finders for your discovery by configuring them as managed processes of DISCO in their respective configuration files. Finders are automatically launched at the appropriate time, provided that CTRL is running.

Each finder must be configured by editing its configuration file. The finders discover the existence of devices and report these devices back to the finders database, but do not discover connections.

Note that the finders database is distinct from the databases that are associated with the individual finders.

The finders are described in the table below, with their executable name and the location of their configuration file. $NCHOME is the environment variable that contains the path to the netcool directory.

Table 69. Description of the finders

Finder	Executable	Configuration file	Description
Ping	ncp_df_ping	$NCHOME/etc/precision/ DiscoPingFinderSchema.cfg $NCHOME/etc/precision/ DiscoPingFinderSeeds.cfg	Makes a simple ICMP echo request for broadcast or multicast addresses, individual IP addresses, or all devices on a subnet.
File	ncp_df_file	$NCHOME/etc/precision/ DiscoFileFinderSchema.cfg $NCHOME/etc/precision/ DiscoFileFinderParseRules.cfg	Parses a file, such as /etc/hosts, to find devices on the network.

Table 69. Description of the finders (continued)

Finder	Executable	Configuration file	Description
Collector	ncp_df_collector	$NCHOME/etc/precision/ DiscoCollectorFinderSchema.cfg $NCHOME/etc/precision/ DiscoCollectorFinderSeeds.cfg	An EMS collector is a software module that retrieves and stores topology data from an Element Management System (EMS). The Collector finder queries a collector and gets a list of IP addresses managed by the EMS associated with that collector.

collectorFinder database

The collectorFinder database defines the operation of the Collector finders.

Description

The collectorFinder database is defined in the `DiscoCollectorFinderSchema.cfg` configuration file. It has the following tables:

- collectorFinder.collectorRules
- collectorFinder.configuration

Related reference:

"DiscoCollectorFinderSeeds.cfg configuration file" on page 56
The `DiscoCollectorFinderSeeds.cfg` configuration file defines how topology data is acquired from Element Management System (EMS) collectors during discovery.

collectorFinder.collectorRules database table

The collectorFinder.collectorRules database table configures the operation of the Collector finder.

Description

You can override some of the settings for particular collectors in the collectorFinder.configuration table. The collectorRules table can contain multiple records.

Schema

The collectorFinder.collectorRules database table schema is described in the following table:

Table 70. collectorFinder.collectorRules database table schema

Column name	Constraints	Data type	Description
`m_Host`		Text	The host address on which the collector is running. This field is NOT NULL only if the collector is running on a different host to Network Manager. This field may be configured for both a discovery and a rediscovery.

Table 70. collectorFinder.collectorRules database table schema (continued)

Column name	Constraints	Data type	Description
`m_Port`	• PRIMARY KEY • NOT NULL	Text	The port on which the collector is listening. If the collector is running on the same host as Network Manager, then this is a Network Manager port. This field may be configured for both a discovery and a rediscovery.
`m_RequestType`		Integer	Flag denoting which topology data to download from the data source. This flag works together with the m_Address and m_NetMask fields. The flag takes the following values: • 0: Rediscover all devices. All devices retrieved by the collector are discovered. The m_Address and m_NetMask fields are ignored. • 1: Rediscover single device. Only one of the devices retrieved by the collector is discovered. The m_Address field specifies the device and the m_NetMask fields is ignored. • 2: Rediscover subnet. One of the subnets retrieved by the collector is discovered. The m_Address field specifies the subnet and the m_NetMask field specifies the subnet mask. This field is configured for a rediscovery only.
`m_DataSourceId`		Integer	Limits rediscovery to a single data source supported by the collector. This field is rarely used as a collector usually only supports a single data source. This field is configured for a rediscovery only.
`m_Address`		Text	Used in conjunction with the m_RequestType and m_NetMask fields when specifying a device or subnet to rediscover. See the entry for m_RequestType for more information. This field is configured for a rediscovery only.
`m_NetMask`		Text	Used in conjunction with the m_RequestType and m_Address fields when specifying a device or subnet to rediscover. See the entry for m_RequestType for more information. This field is configured for a rediscovery only.

Table 70. collectorFinder.collectorRules database table schema (continued)

Column name	Constraints	Data type	Description
m_NumRetries		Integer	Number of retries to issue an RPC XML request to the collector. Setting this field is optional. If set, this field overrides the default specified in the collectorFinder.configuration table. This field may be configured for both a discovery and a rediscovery.

collectorFinder.configuration database table

The collectorFinder.configuration table specifies the general rules of the Element Management System (EMS) collector methodology and must only contain one record.

Schema

The collectorFinder.configuration database table schema is described in the following table:

Table 71. collectorFinder.configuration database table schema

Column name	Constraints	Data type	Description
m_NumThreads		Integer	The number of threads to be used by the Collector finder.
m_TimeOut		Integer	The maximum time to wait for a reply from a collector (the timeout).
m_NumRetries		Integer	The number of times to issue an XML-RPC request to a collector.
m_MaxResponseSize		Integer	The maximum size for an XML-RPC response in bytes. **Note:** The default maximum response size might be too small when running a Collector-based discovery against Collectors that result in very large responses. In such cases, increase the maximum response size. To increase the maximum response size, set the **m_MaxResponseSize** parameter to a higher value. Make sure you set the same value for **m_MaxResponseSize** in both of the following files: • NCHOME/etc/precision/DiscoCollectorFinderSchema.cfg • NCHOME/etc/precision/DiscoXmlRpcHelperSchema.cfg

fileFinder database

The fileFinder database defines the operation of the File finder.

Description

The fileFinder database is defined in the DiscoFileFinderParseRules.cfg file. It has the following tables:

- fileFinder.configuration
- fileFinder.parseRules

Related reference:

"DiscoFileFinderParseRules.cfg configuration file" on page 58
The DiscoFileFinderParseRules.cfg file can be used to specify the files to be parsed for a list of IP addresses of devices that exist on the network.

fileFinder.configuration database table

You can configure the File finder with the fileFinder.configuration table, which specifies the number of threads to be used by the finder.

Schema

The fileFinder.configuration database table is described in the following table.

Table 72. fileFinder.configuration database table schema

Column name	Constraints	Data type	Description
`m_NumThreads`	NOT NULL	Integer	The number of threads to be used by the File finder.

fileFinder.parseRules database table

By configuring inserts into the fileFinder.parseRules table, you can specify the files to be parsed for a list of IP addresses of devices on the network.

Description

The fileFinder.parseRules table specifies the rules for file parsing.

A typical file that you would parse, for example, is the `/etc/hosts` file on the machine running DISCO. You can also seed the discovery by parsing the `/etc/defaultrouter` file.

Schema

The fileFinder.parseRules database table schema is described in the following table:

Table 73. fileFinder.parseRules database table schema

Column name	Constraints	Data type	Description
`m_FileName`	• NOT NULL • UNIQUE	Text	The unique full path and filename of the file to be parsed, for example, `/etc/hosts`.

Table 73. fileFinder.parseRules database table schema (continued)

Column name	Constraints	Data type	Description
m_Delimiter		Text	The delimiter that separates the data fields in the file. Regular pattern matching expressions are also accepted as valid delimiters. **Note:** \t is not supported as a valid value for the <tab> character.
m_ColDefs		List of atoms	A list of rules that specify which variables to extract and the columns from which to get them.

pingFinder database

The pingFinder database defines the operation of the Ping finder.

Description

The pingFinder database is defined in the DiscoPingFinderSeeds.cfg file. It has the following tables:

- pingFinder.configuration
- pingFinder.pingFilter
- pingFinder.pingRules
- pingFinder.scope

Related reference:

"DiscoPingFinderSeeds.cfg configuration file" on page 61
The DiscoPingFinderSeeds.cfg configuration file is used for seeding the Ping finder and restricting device detection.

pingFinder.configuration database table

The pingFinder.configuration table specifies the general rules of the ping methodology. The table must contain only one record.

Description

The pingFinder.configuration table allows you to configure the way devices are pinged, including enabling broadcast or multicast pinging. Although pinging of broadcast/multicast addresses allows devices to be discovered more quickly than other detection methods, it is sometimes less desirable to do so under certain network conditions, such as when the network is heavily congested. In general, you would ping broadcast addresses on an unknown sparsely populated network. You must only ping multicast addresses where they have been set up on the network.

Schema

The pingFinder.configuration database table schema is described in the following table:

Table 74. pingFinder.configuration database table schema

Column name	Data type	Description
m_NumThreads	Integer	The number of threads to be used by the Ping finder.

Table 74. pingFinder.configuration database table schema (continued)

Column name	Data type	Description
m_TimeOut	Integer	The maximum time to wait for a reply from a pinged address (the timeout).
m_InterPingTime	Integer	The interval between pinging the addresses in a subnet.
m_NumRetries	Integer	The number of times a device is to be re-pinged.
m_Broadcast	Integer	Flag used to enable or disable broadcast address pinging: • 1: Enable • 0: Disable
m_Multicast	Integer	Flag used to enable or disable multicast address pinging: • 1: Enable • 0: Disable

pingFinder.pingFilter database table

The pingFinder.pingFilter table can be used to exclude particular devices or subnets from being pinged by the Ping finder.

Description

You may wish to exclude certain interfaces, such as ISDN and modem interfaces, because pinging these interfaces generates phone calls, which costs money. If you configure the Ping finder to use both the scope.zones table and the pingFinder.pingFilter table, the Ping finder pings those devices or subnets it has been seeded with if they are within either the discovery scope or the Ping finder scope.

Schema

The pingFinder.pingFilter database table schema is described in the following table:

Table 75. pingFinder.pingFilter database table schema

Column name	Constraints	Data type	Description
m_Protocol	• PRIMARY KEY • NOT NULL • Externally defined netProtocol data type	Integer	An integer representation of the network protocol used by the presently defined Ping finder zone. Currently only IP is supported: • 0: Undefined • 1: IP
m_Action	• NOT NULL • Externally defined netProtocol data type	Integer	Action to perform for current zone: • 0: Undefined • 1: Include • 2: Exclude
m_Zones		List of type zone	A list of varbinds (name=value) that define the present zone.

Table 75. pingFinder.pingFilter database table schema (continued)

Column name	Constraints	Data type	Description
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

pingFinder.pingRules database table

The pingFinder.pingRules table specifies the different addresses and subnets to be pinged by the Ping finder.

Description

The pingRules table can contain multiple records.

Schema

The pingFinder.pingRules table is described in the following table.

Table 76. pingFinder.pingRules database table schema

Column name	Constraints	Data type	Description
m_Address	• PRIMARY KEY • NOT NULL	Text	The address to ping.
m_RequestType		Integer	Flag denoting address type: • 1: Individual • 2: Subnet
m_NetMask		Text	The subnet mask. If a value is specified for this field, it automatically implies that the address is a subnet address.
m_TimeOut		Integer	Maximum time to wait for response. This value overrides the default timeout specified in the configuration table.
m_NumRetries		Integer	Maximum number of times to reattempt the ping. This value overrides the default value.

pingFinder.scope database table

The pingFinder.scope table defines the scope of the Ping finder.

Description

You can use the pingFinder.scope table to configure the way the Ping finder checks whether it is allowed to ping a particular device. You can exclude particular devices or subnets from being pinged by the Ping finder.

Schema

The pingFinder.scope database table schema is described in the following table:

Table 77. pingFinder.scope database table schema

Column name	Constraints	Data type	Description
m_UseScope		Integer	Flag denoting whether or not to use the entries in the scope.zones table when deciding which devices to ping: • 0: The Ping finder ignores the scope.zones table when deciding which devices to ping. • 1: This is the default value. The Ping finder uses the scope.zones table to check which devices can be pinged. If you are performing an unscoped discovery, that is, a discovery without any entries in the scope.zones table, then it is preferable to set m_UseScope to zero to reduce processing load.
m_UsePingEntries		Integer	Flag denoting whether or not to use the entries in the pingFinder.pingFilter table when deciding which devices to ping: • 0: This is the default value. The Ping finder ignores any entries in the pingFinder.pingFilter table when deciding which devices can be pinged. • 1: The Ping finder checks the pingFinder.pingFilter table before it pings a particular device to see if the device can be pinged.

The Helper Server databases

When the Helper Server starts, it creates a database for each helper that is to be run.

Tip: It is good practice to configure the Helper Server to start automatically by making the appropriate OQL insertion into the services.inTray table of CTRL. Alternatively, you can start the Helper Server manually with the ncp_d_helpserv command on the command line.

Related reference:

"DiscoHelperServerSchema.cfg configuration file" on page 60
The DiscoHelperServerSchema.cfg configuration file defines the contents of the several helper databases.

The ARPhelper database

The ARPHelper database stores information about the requests the ARP helper makes from the network. It is defined in $NCHOME/etc/precision/ DiscoHelperServerSchema.cfg, and its fully qualified database table names are: ARPHelper.ARPHelperTable; ARPHelper.ARPHelperConfig.

The `ARPHelperTable` database table, described in Table 78, configures the general operation of the ARP helper.

Table 78. ARPHelper.ARPHelperTable database table schema

Column name	Constraints	Data type	Description
`RivHelperRequestReplyKey`	• PRIMARY KE • NOT NULL • UNIQUE	Text	A unique key interface to the databases of the Helper Server for Reply requests.
`RivHelperRequestGetKey`	NOT NULL	Text	A key interface to the databases of the Helper Server for Get requests.
`RivHelperDbTimeToDie`		Long64	Indicates how long the requested information is to live within the Helper Server.
`m_HostIp`	NOT NULL	Text	IP address of the device to interrogate.
`m_HostSubnet`		Text	Subnet of the host device to be interrogated.
`m_HostMask`		Text	The subnet mask of the host device to be interrogated.
`m_HostMac`		Text	The physical address of the device (MAC address).

The `ARPHelperConfig` table, described in Table 79, contains configuration information for the ARP helper.

Table 79. ARPHelper.ARPHelperConfig database table schema

Column name	Constraints	Data type	Description
`m_HelperDbTimeout`	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires in the absence of any activity.
`m_HelperReqTimeout`		Long64	The helper request timeout, that is, how long before each request expires.
`m_HelperStartupTimeout`		Long64	The default helper startup timeout, that is, the maximum time to wait for a helper to start up when requested.

Table 79. ARPHelper.ARPHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
m_HelperDoWeQuery		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: (0) Do not use cache (1) Use cache
m_HelperDoQueryVBs Optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.
m_HelperDoNotQueryVBs Optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides the value specified in m_HelperDoWeQuery.
m_HelperDoWeStore		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: (0) Do not store replies in database (1) Store replies in database
m_HelperDoStoreVBs Optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides the value of m_HelperDoWeStore.
m_HelperDoNotStoreVBs Optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides the value of m_HelperDoWeStore.
m_HelperDebugLevel Optional		Integer	Sets the debug level of the helper, printing to m_HelperLogfile.
m_HelperLogfile Optional		Text	The full path and file for the logfile of the current helper.

The m_HelperDoWeQuery and m_HelperDoWeStore fields each have two related optional fields. A record entered into either m_HelperDoWeQuery or m_HelperDoWeStore is the default setting to which the helper responds if no records are entered into the optional fields. However, a record entered into either of the related optional fields overrides the default setting.

For example, if m_HelperDoWeQuery is set to query the network rather than the cache (that is, m_HelperDoWeQuery=0) and if an IP address of 192.168.0.1 is specified in m_HelperDoQueryVBs, then a request record where `m_IpAddress = 192.168.0.1` results in the cache being queried rather than the network. The network is only queried if the information is not currently held in the cache.

ARPhelper database configuration

The following example insert gives a typical ARP helper configuration.

```
insert into ARPHelper.ARPHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200, 1200, 90, 0, 0
);
```

DNS helper database schema

The DNSHelper database is defined in $NCHOME/etc/precision/ DiscoHelperServerSchema.cfg. Its fully qualified database table names are: DNSHelper.DNSHelperTable; DNSHelper.DNSHelperConfig

The DNSHelper database table stores information about the requests that the ARP helper makes from the network.

Table 80. DNSHelper.DNSHelperTable database table schema

Column name	Constraints	Data type	Description
RivHelperRequestReplyKey	• PRIMARY KEY • NOT NULL • UNIQUE	Text	A unique key for Reply requests.
RivHelperRequestGetKey	NOT NULL	Text	A key for Get requests.
RivHelperDbTimeToDie		Long64	How long the requested information is to live within the Helper Server.
m_HostName		Text	The host name for this IP address.
m_HostIp		Text	The IP addresses for this host.
RivHelperRequestOutput		Atom	The response data.

The DNSHelperConfig table holds configuration information for the DNS helper.

Table 81. DNSHelper.DNSHelperConfig database table schema

Column name	Constraints	Data type	Description
m_HelperDbTimeout	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires.
m_HelperReqTimeout		Long64	The helper request timeout, that is, how long before each request expires.

Table 81. DNSHelper.DNSHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
`m_HelperStartupTimeout`		Long64	The default helper start-up timeout, that is, the maximum time to wait for a helper to start up when requested.
`m_HelperDoWeQuery`		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: • 0: Do not use cache • 1: Use cache
`m_HelperDoNotQueryVBs` optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides the value of m_HelperDoWeQuery.
`m_HelperDoQueryVBs` optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.
`m_HelperDoWeStore`		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: • 0: Do not store replies in database • 1: Store replies in database
`m_HelperDoStoreVBs` optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides m_HelperDoWeStore.
`m_HelperDoNotStoreVBs` optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides m_HelperDoWeStore.
`m_HelperDebugLevel` optional		Integer	Sets the debug level of the helper, printing to m_Logfile.
`m_HelperLogfile` optional		Text	The full path and file for the logfile of the current helper.

DNS helper database configuration

The following example insert shows a typical DNS helper configuration.

```
insert into DNSHelper.DNSHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200, 1200, 90, 0, 0
);
```

Ping helper database schema

The Ping helper database is defined in `$NCHOME/etc/precision/DiscoHelperServerSchema.cfg`. Its fully qualified database table names are: PingHelper.PingHelperTable; PingHelper.PingHelperConfig; pingHelper.configuration

The schema of the PingHelper.PingHelperTable database table is described in Table 82.

Table 82. PingHelper.PingHelperTable database table schema

Column name	Constraints	Data type	Description
`RivHelperRequestReplyKey`	• PRIMARY KEY • NOT NULL • UNIQUE	Text	A key interface to the databases of the Helper Server for Reply requests.
`RivHelperRequestGetKey`	NOT NULL	Text	A key interface to the databases of the Helper Server for Get requests.
`RivHelperDbTimeToDie`		Long64	How long the requested information is to live within the Helper Server.
`m_HostIp`		Atom	IP address to ping.
`m_HostSubnet`		Text	Subnet of the IP address to ping.
`m_HostMask`		Text	The subnet mask of the address to ping.
`m_PingRequestType`		Integer	The type of ping request: • 1: Individual address • 2: Subnet
`m_PingResponseType`		Integer	Type of reply to the ping.
`m_PingRetries`		Integer	Number of retries for the ping.
`m_PingTimeout`		Integer	Maximum time to wait for reply.
`RivHelperRequestOutput`		Atom	The response data.

The schema of the PingHelper.PingHelperConfig database table is described in Table 83.

Table 83. PingHelper.PingHelperConfig database table schema

Column name	Constraints	Data type	Description
`m_HelperDbTimeout`	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires.
`m_HelperReqTimeout`		Long64	The helper request timeout that is, how long before each request expires.
`m_HelperStartupTimeout`		Long64	The default helper startup timeout, that is, the maximum time to wait for a helper to start up when requested to.
`m_HelperDoWeQuery`		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: • 0: Do not use cache • 1: Use cache
`m_HelperDoNotQueryVBs` optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides m_HelperDoWeQuery.
`m_HelperDoQueryVBs` optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.
`m_HelperDoWeStore`		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: • 0: Do not store replies in database • 1: Store replies in database
`m_HelperDoStoreVBs` optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides m_HelperDoWeStore.

Table 83. PingHelper.PingHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
m_HelperDoNotStoreVBs optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides m_HelperDoWeStore.
m_HelperDebugLevel optional		Integer	Sets the debug level of the helper, printing to the file specified in m_HelperLogfile.
m_HelperLogFile optional		Text	The full path and file for the logfile of the current helper.

The schema of the pingHelper.configuration database table is described in Table 84. It must contain only one record.

Although pinging broadcast and multicast addresses allows devices to be discovered quicker than other detection methods, it is not advisable to do so under certain network conditions; for instance, when the network is heavily congested.

Table 84. pingHelper.configuration database table schema

Column name	Constraints	Data type	Description
m_NumThreads		Integer	The number of threads to be used by the helper.
m_TimeOut		Integer	The maximum time to wait for a reply from a pinged address, in milliseconds. If you are running the TraceRoute agent you may need to increase this value, depending on network conditions.
m_NumRetries		Integer	The number of times a device is to be re-pinged.
m_InterPingTime		Integer	The time interval in milliseconds between successive ping attempts of subnet addresses.
m_Broadcast		Integer	Flag used to enable or disable broadcast address pinging: • (1) Enable • (0) Disable
m_Multicast		Integer	Flag used to enable or disable multicast address pinging: • (1) Enable • (0) Disable

PING helper database configuration

The following insert provides a typical example configuration of the PingHelper database.

```
insert into PingHelper.PingHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200, 1200, 90, 0, 0
);
```

SNMP helper database schema

The SnmpHelper database is defined in $NCHOME/etc/ precision/ DiscoHelperServerSchema.cfg. Its fully qualified database table names are: SnmpHelper.SnmpHelperTable; SnmpHelper.SnmpHelperConfig.

The schema of the SNMPHelperTable database table is described in Table 85.

Table 85. SnmpHelper.SnmpHelperTable database table schema

Column name	Constraints	Data type	Description
RivHelperRequestReplyKey	• PRIMARY KEY • NOT NULL • UNIQUE	Text	A key interface to the databases of the Helper Server for Reply requests.
RivHelperRequestGetKey	NOT NULL	Text	A key interface to the databases of the Helper Server for Get requests.
RivHelperDbTimeToDie		Long64	How long the requested information is to live within the Helper Server.
m_HostIp	NOT NULL	Text	IP address of the device to interrogate.
m_CommunitySuffix		Text	The suffix to the community string.
m_OID	NOT NULL	Atom	Object ID for the Get request.
m_SnmpIndex		Atom	The index of the Get request (if it is a Get request).
m_RequestType		Integer	Type of request: • 0: Get • 1: GetNext • 2: GetBulk

Table 85. SnmpHelper.SnmpHelperTable database table schema (continued)

Column name	Constraints	Data type	Description
RivHelperRequestOutput		Atom	The response data.

The schema of the SNMPHelperConfig database table is described in Table 86.

Table 86. SnmpHelper.SnmpHelperConfig database table schema

Column name	Constraints	Data type	Description
m_HelperDbTimeout	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires.
m_HelperReqTimeout		Long64	The helper request timeout, that is, how long before each request expires.
m_HelperStartupTimeout		Long64	The default helper startup timeout, that is, the maximum time to wait for a helper to start up when requested to.
m_HelperDoWeQuery		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: • 0: Do not use cache • 1: Use cache
m_HelperDoNotQueryVBs optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides m_HelperDoWeQuery.
m_HelperDoQueryVBs optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.
m_HelperDoWeStore		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: • 0: Do not store replies in database • 1: Store replies in database

Table 86. SnmpHelper.SnmpHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
m_HelperDoStoreVBs optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides m_HelperDoWeStore.
m_HelperDoNotStoreVBs optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides m_HelperDoWeStore.
m_HelperDebugLevel optional		Integer	Sets the debug level of the helper, printing to m_HelperLogfile.
m_HelperLogfile optional		Text	The full path and file for the logfile of the current helper.

SNMP helper database configuration

The following insert provides an example configuration of the SNMP helper database.

```
insert into SnmpHelper.SnmpHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200, 1200, 90, 0, 0
);
```

Telnet helper database schema

The TelnetHelper database is defined in $NCHOME/etc/ precision/ DiscoHelperServerSchema.cfg. Its fully qualified database table names are: TelnetHelper.TelnetHelperTable; TelnetHelper.TelnetHelperConfig.

The TelnetHelperTable database table schema is described in Table 87.

Table 87. TelnetHelper.TelnetHelperTable database table schema

Column name	Constraints	Data type	Description
RivHelperRequestReplyKey	• PRIMARY KEY • NOT NULL • UNIQUE	Text	A unique request reply key interface to the databases of the Helper Server.

Table 87. TelnetHelper.TelnetHelperTable database table schema (continued)

Column name	Constraints	Data type	Description
RivHelperRequestGetKey	NOT NULL	Text	A request get key interface to the databases of the Helper Server.
RivHelperDbTimeToDie		Long64	How long the requested information is to live within the Helper Server.
m_HostIp	NOT NULL	Text	IP address of the device to interrogate.
m_TelnetCommand		Text	The Telnet command.
RivHelperRequestOutput		Atom	The response data.

Table 88 gives the schema of the TelnetHelperConfig table.

Table 88. TelnetHelper.TelnetHelperConfig database table schema

Column name	Constraints	Data type	Description
m_HelperDbTimeout	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires.
m_HelperReqTimeout		Long64	The helper request timeout, that is, how long before each request expires.
m_HelperStartupTimeout		Long64	The default helper start-up timeout, that is, the maximum time to wait for a helper to start up when requested.
m_HelperDoWeQuery		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: • 0: Do not use cache • 1: Use cache
m_HelperDoNotQueryVBs optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides m_HelperDoWeQuery.
m_HelperDoQueryVBs optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.

Table 88. TelnetHelper.TelnetHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
m_HelperDoWeStore		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: • 0: Do not store replies in database • 1: Store replies in database
m_HelperDoStoreVBs optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides m_HelperDoWeStore.
m_HelperDoNotStoreVBs optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides m_HelperDoWeStore.
m_HelperDebugLevel optional		Integer	Sets the debug level of the helper, printing to m_HelperLogfile.
m_HelperLogfile optional		Text	The full path and file for the logfile of the current helper.

Telnet helper database configuration

The following example insert gives a typical configuration of the Telnet helper database.

```
insert into TelnetHelper.TelnetHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200, 1200, 90, 0, 0
);
```

XMLRPC helper database schema

The XmlRpcHelper helper database is defined in $NCHOME/etc/precision/DiscoHelperServerSchema.cfg. Its fully qualified database table names are: XmlRpcHelper.XmlRpcHelperTable; XmlRpcHelper.XmlRpcHelperConfig.

The schema of the XmlRpcHelper.XmlRpcHelperTable database table is described in Table 89 on page 246.

Table 89. XmlRpcHelper.XmlRpcHelperTable database table schema

Column name	Constraints	Data type	Description
RivHelperRequestReplyKey	• PRIMARY KEY • NOT NULL • UNIQUE	Text	A key interface to the databases of the Helper Server for Reply requests.
RivHelperRequestGetKey	NOT NULL	Text	A key interface to the databases of the Helper Server for Get requests.
RivHelperDbTimeToDie		Text	How long the requested information is to live within the Helper Server.
m_port		Atom	Port of physical device.
m_DataSourceId		Integer	Data source of interest.
m_MethodCalled		Text	Method called.
m_MethodSignature		Integer	Method signature.
RivHelperRequestOutput		Atom	Response data.

The schema of the XmlRpcHelper.XmlRpcHelperConfig database table is described in Table 90.

Table 90. XmlRpcHelper.XmlRpcHelperConfig database table schema

Column name	Constraints	Data type	Description
m_HelperDbTimeout	UNIQUE	Long64	The helper database timeout, that is, how long before the database expires.
m_HelperReqTimeout		Long64	The helper request timeout that is, how long before each request expires.
m_HelperStartupTimeout		Long64	The default helper startup timeout, that is, the maximum time to wait for a helper to start up when requested to.
m_HelperDoWeQuery		Integer	Indicates whether the Helper Server queries its database or whether it queries the network using a helper: • 0: Do not use cache • 1: Use cache
m_HelperDoNotQueryVBs optional		Object type varbinds	List of helper inputs that do not query the database. This field overrides m_HelperDoWeQuery.
m_HelperDoQueryVBs optional		Object type varbinds	List of helper inputs that always query the database before querying the network. If the item is found in the database then the network is not queried.

Table 90. XmlRpcHelper.XmlRpcHelperConfig database table schema (continued)

Column name	Constraints	Data type	Description
`m_HelperDoWeStore`		Integer	Indicates whether the Helper Server stores any replies from the helpers in its database: • 0: Do not store replies in database • 1: Store replies in database
`m_HelperDoStoreVBs` optional		Object type varbinds	List of helper inputs that always store data in the Helper Server database. This field overrides m_HelperDoWeStore.
`m_HelperDoNotStoreVBs` optional		Object type varbinds	List of helper inputs that never store data in the Helper Server databases. This field overrides m_HelperDoWeStore.
`m_HelperDebugLevel` optional		Integer	Sets the debug level of the helper, printing to the file specified in m_HelperLogfile.
`m_HelperLogFile` optional		Text	The full path and file for the logfile of the current helper.

XMLRPC helper database configuration

The following insert provides a typical example configuration of the XmlRpcHelper database. This insert specifies the following settings:

- Helper database expires after 3 days.
- Each helper database request timeout expires after 20 minutes.
- Maximum time to wait for a helper to start up when requested is 90 seconds.
- Helper Server does not query its database.
- Helper Server does not store any replies from the helpers in its database.

```
insert into XmlRpcHelper.XmlRpcHelperConfig
(
        m_HelperDbTimeout,
        m_HelperReqTimeout,
        m_HelperStartupTimeout,
        m_HelperDoWeQuery,
        m_HelperDoWeStore
)
values
(
        259200,
 1200,
 90,
 0,
0
);
```

Individual helpers databases

In addition to the DiscoHelperServerSchema.cfg, each of the helpers has an associated configuration file that governs the behavior of the helper. The following topics describe the databases for the individual configuration files.

The ARP helper database

The ARP helper database is defined by the DiscoARPHelperSchema.cfg configuration file Its fully qualified database table name is ARPHelper.configuration.

The ARPHelper.configuration database, described in Table 91, defines the number of threads the helper uses.

Table 91. ARPHelper.configuration database table schema

Column Name	Constraints	Data type	Description
`m_NumThreads`	None	Integer	The number of threads to be used by the helper.

Related reference:

"DiscoARPHelperSchema.cfg configuration file" on page 55
The DiscoARPHelperSchema.cfg configuration file performs IP address to MAC address resolution.

The DNS helper database

The DNS helper database is defined by the DiscoDNSHelperSchema.cfg configuration file. Its fully qualified database table names are: DNSHelper.configuration; DNShelper.methods.

The `DNSHelper.configuration` table, described must contain only one record.

Table 92. DNSHelper.configuration database table schema

Column name	Constraints	Data type	Description
`m_NumThreads`		Integer	The number of threads to be used by the helper.
`m_MethodList`		List of text	An ordered list of the methods for name retrieval.
`m_TimeOut`		Integer	The maximum time to wait for a response from a device (seconds).

Table 93. DNShelper.methods database table schema

Column name	Constraints	Data type	Description
`m_MethodName`	• PRIMARY KEY • NOT NULL • UNIQUE	Text	The name of the method.
`m_MethodType`		Integer	The type of the method: • 0: System • 1: DNS • 2: File

Table 93. DNShelper.methods database table schema (continued)

Column name	Constraints	Data type	Description
m_NameServerAddr		Text	The IP address of the DNS server (specified as a text string). If no value is specified, /etc/resolv.conf is read.
m_NameDomain		Text	Domain name; for example, abcd.com.
m_NameDomainList		Text	Contains a list of expected domain suffixes. If you expect the discovery to return some or all devices names with domain suffixes already appended, then you can specify a list of expected domain suffixes in this column. **Note:** The domain suffix value specified in m_NameDomain is not appended to any device names returned by the discovery that have any of the suffixes listed in m_NameDomainList.
m_FileName		Text	The filename, if appropriate.
m_FileOrder		Integer	The order of the files: • 0: Name first, then IP address • 1: IP address, then name
m_TimeOut		Integer	Time out for the request in seconds.

Related reference:

"DiscoDNSHelperSchema.cfg configuration file" on page 56
The DiscoDNSHelperSchema.cfg configuration file defines access to DNS, which enables the discovery to do domain name lookups, by configuring the DNS helper.

The Ping helper database

The Ping helper database is defined by the DiscoPingHelperSchema.cfg configuration file. Its fully qualified database table name is pingHelper.configuration.

The schema of the pingHelper.configuration database table is described in Table 84 on page 240. It must contain only one record.

Although pinging broadcast and multicast addresses allows devices to be discovered quicker than other detection methods, it is not advised to do so under certain network conditions; for instance, when the network is heavily congested.

Table 94. pingHelper.configuration database table schema

Column name	Constraints	Data type	Description
m_NumThreads		Integer	The number of threads to be used by the helper.

Table 94. pingHelper.configuration database table schema (continued)

Column name	Constraints	Data type	Description
m_TimeOut		Integer	The maximum time to wait for a reply from a pinged address, in milliseconds. If you are running the TraceRoute agent you may need to increase this value, depending on network conditions.
m_NumRetries		Integer	The number of times a device is to be re-pinged.
m_InterPingTime		Integer	The time interval in milliseconds between successive ping attempts of subnet addresses.
m_Broadcast		Integer	Flag used to enable or disable broadcast address pinging: • (1) Enable • (0) Disable
m_Multicast		Integer	Flag used to enable or disable multicast address pinging: • (1) Enable • (0) Disable

Related reference:

"DiscoPingHelperSchema.cfg configuration file" on page 62
The DiscoPingHelperSchema.cfg configuration file defines how devices are to be pinged.

The SNMP helper database

The SNMP helper database is defined by the DiscoSnmpHelperSchema.cfg configuration file. Its fully qualified database table name is snmpHelper.configuration.

The SNMP helper database consists of the snmpHelper.configuration table, described in Table 95, which must contain only one record.

Table 95. snmpHelper.configuration database table schema

Column name	Constraints	Data type	Description
m_NumThreads	None	Integer	The number of threads to be used by the helper.
m_TimeOut	None	Integer	The maximum time to wait for a reply from a device, in milliseconds.
m_NumRetries	None	Integer	The number of attempts to retrieve SNMP variable(s) from a device.

Related reference:

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

The Telnet helper database

The Telnet helper database is defined by the DiscoTelnetHelperSchema.cfg configuration file. Its fully qualified database table names are: telnetHelper.configuration; telnetHelper.deviceConfig.

The telnetHelper.configuration table specifies the general rues of receiving information from remote devices.

Table 96. telnetHelper.configuration database table schema

Column name	Constraints	Data type	Description
`m_NumThreads`		Integer	The number of threads to be used by the helper. If you change this value, be sure that your system is configured to allow at least this number of concurrent Telnet sessions.
`m_TimeOut`		Integer	The maximum time to wait for access to a device (milliseconds).
`m_Retries`		Integer	The number of times to retry the device.

The telnetHelper.deviceConfig table sets device-specific configuration options.

Table 97. telnetHelper.deviceConfig database table schema

Column name	Constraints	Data type	Description
`m_SysObjectId` optional		Text	The `sysObjectID` MIB variable to match for this configuration entry. The entry with the longest OID match will be the entry used. For example, if you specify a value of 1.3.6.1.4.1.9.1 then all devices with OIDs of the form 1.3.6.1.4.1.9.1.* will be matched. Cisco IOS devices have OIDs of the form 1.3.6.1.4.1.9.1.*. This field is ignored if m_IpOrSubNet is specified.
`m_IpOrSubNet`		Text	The IP or fully qualified subnet address of the device corresponding to a particular configuration. If this is not specified, the configuration is used as the default subnet address.
`m_NetMaskBits`		Integer	The number of most significant bits in the netmask. This number must be specified if `m_IpOrSubNet` is specified.
`m_PageLengthCmd`		Text	The command to issue in order to set the output page length.

Table 97. telnetHelper.deviceConfig database table schema (continued)

Column name	Constraints	Data type	Description
m_PageLength		Integer	The output page length size. This is set to 0 by default; that is, no paging. If you set a page length size, you must also insert a value into the m_PageLengthCmd column in order to set a page length command.
m_ContinueMsg		Text	The expected prompt from the remote device between paged output; for example, "Do you want to continue". Regular expressions are valid entries.
m_ContinueCmd		Text	The response to send to the remote device in order for it to continue the paged output. This is usually set to "y". You must take care setting this value, as some devices require a carriage return after the command and some do not. For maximum flexibility, a return is not added by default. It must be specified explicitly using a trailing Ctrl-M in the string.
m_TransmissionDelay		Integer	This option allows you to customize the delay used by ncp_dh_telnet when transmitting data to a device. This may be useful if data loss or device issues occur when using the default transmission delay setting.

Related reference:

"DiscoTelnetHelperSchema.cfg configuration file" on page 71
The DiscoTelnetHelperSchema.cfg configuration file defines the operation of the Telnet helper, which returns the results of a Telnet operation into a specified device.

The XMLRPC helper database

The XMLRPC helper database is defined by the DiscoXmlRpcHelperSchema.cfg configuration file. Its fully qualified database table name is xmlRpcHelper.configuration.

The schema of the xmlRpcHelper.configuration database table is described in Table 98. It must contain only one record.

Table 98. xmlRpcHelper.configuration database table schema

Column name	Constraints	Data type	Description
m_NumThreads	None	Integer	The number of threads to be used by the helper.

Table 98. xmlRpcHelper.configuration database table schema (continued)

Column name	Constraints	Data type	Description
m_TimeOut	None	Integer	The maximum time to wait for a reply from an EMS collector, in milliseconds. If you are running the TraceRoute agent you may need to increase this value, depending on network conditions.

Related reference:

"DiscoXmlRpcHelperSchema.cfg configuration file" on page 74
The DiscoXmlRpcHelperSchema.cfg configuration file can be used to configure the XML-RPC helper, which enables Network Manager to communicate with EMS collectors using the XML-RPC interface.

Tracking discovery databases

During the discovery process, the discovery engine, ncp_disco, records every element discovered in the network, whether it has been processed or not. The instrumentation and translations databases are used for this purpose. These databases can be interrogated at any time to view the number of device types and categories that have been discovered.

The translations, instrumentation, and workingEntities databases record the known network entities and technologies, and can be used to track the progress of the discovery.

translations database

The translations database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. It has several fully qualified database table names.

The fully qualified database table name for the translations database are:

- translations.ipToBaseName
- translations.vlans
- translations.NAT
- translations.NATtemp
- translations.NATAddressSpaceIds

translations.ipToBaseName table

The ipToBaseName table is a registry of discovered devices and the IP addresses associated with those devices.

When a device has multiple interfaces, and therefore multiple IP addresses, the Associated Address agent downloads all the associated addresses, stores them in the ipToBaseName table and allows the appropriate discovery agents to discover the device. Any subsequent attempt to discover the device by means of another of its IP addresses is halted when the Associated Address agent checks the ipToBaseName table, that is, before the device details are passed to the appropriate discovery agent.

Table 99. translations.ipToBaseName database table schema

Column name	Constraints	Data type	Description
m_BaseName	NOT NULL	Text	Base name of the discovered entity.

Table 99. translations.ipToBaseName database table schema (continued)

Column name	Constraints	Data type	Description
m_BaseAddress	NOT NULL	Text	Base address of the discovered entity.
m_WorkAddress	NOT NULL	Text	The address that was used for data retrieval.
m_IpAddress	NOT NULL	Text	IP address of the entity.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
m_InScope		Boolean integer	Indicates whether the value of the field m_IpAddress is in scope.
m_Protocol	NOT NULL	Integer	Protocol for this address. This field can take the following values: • 1: *IPv4* • 3: *IPv6*
m_IsManagementIP		Boolean integer	Indicates whether this is a management IP address.
m_IsOutOfBand		Boolean integer	Indicates whether this is an out of band address.
m_Name		Text	Name of interface with IP if known.

translations.vlans table

The vlans table holds a list of devices that are part of Virtual Local Area Networks (VLANs). Each record in the vlans table maps the device to the VLAN to which it belongs.

Table 100. translations.vlans database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL	Text	The name of the device associated with this entry.
m_VlanID	• PRIMARY KEY • NOT NULL	Text	The VLAN identifier on the device.
m_Subnet		Text	The subnet with which the VLAN appears to be associated.
m_NetMask		Text	The subnet mask.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

translations.NAT table

The NAT table is used to hold static NAT mappings. The mapped devices are discovered even if they are outside the scope of the discovery.

Table 101. translations.NAT database table schema

Column name	Constraints	Data type	Description
m_OutsideGlobalAddr	• PRIMARY KEY • NOT NULL	Text	The public address.
m_InsideLocalAddr	NOT NULL	Text	The private address.
m_InsideGlobalAddr		Text	This column is currently not used.
m_OutsideLocalAddr		Text	This column is currently not used.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

translations.NATtemp

The NATtemp table is used to hold NAT mappings from a particular NAT gateway. This enables the discovery process to compare the old and new NAT mappings and initiate a partial or full rediscovery if necessary.

Table 102. translations.NATtemp database table schema

Column name	Constraints	Data type	Description
m_OutsideAddr	• PRIMARY KEY • NOT NULL	Text	The public address of the device.
m_InsideAddr	NOT NULL	Text	The private address of the device.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.

translations.NATAddressSpaceIds table

The NATAddressSpaceIds table is used to identify the IP addresses of NAT gateways and specify an address-space identifier for each one.

Table 103. translations.NATAddressSpaceIds database table schema

Column name	Constraints	Data type	Description
m_NATGatewayIP	• PRIMARY KEY • NOT NULL	Text	The IP address of the gateway.

Table 103. translations.NATAddressSpaceIds database table schema (continued)

Column name	Constraints	Data type	Description
m_AddressSpaceId		Text	The address space identifier to be used for all devices in the NAT domain belonging to the gateway whose IP address is specified in m_NATGatewayIP.

Related tasks:

"Defining address spaces for NAT gateways" on page 120
To specify the IP address of your NAT gateways and the address space identifier you want to use for each associated NAT domain, edit DiscoConfig.cfg to create or amend an insert into translations.NATAddressSpaceIds.

instrumentation database schema

The instrumentation database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. It lists discovered devices grouped by technology. You can do OQL queries to retrieve the names of all discovered subnets, VLANs, Frame Relay devices, and so on.

The fully qualified database table names for the instrumentation database are:

- instrumentation.ipAddresses
- instrumentation.name
- instrumentation.subNet
- instrumentation.vlan
- instrumentation.frameRelay
- instrumentation.ciscoFrameRelay
- instrumentation.hsrp
- instrumentation.pnniPeerGroup
- instrumentation.fddi

instrumentation.ipAddresses table

The ipAddresses table contains a record of the unique IP addresses discovered in the network.

Table 104. instrumentation.ipAddresses database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	• PRIMARY KEY • NOT NULL • UNIQUE	Text	The IP address of a discovered network entity.

instrumentation.name table

The name table contains a record of the unique name of every discovered device.

Table 105. instrumentation.name database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	The name of a discovered network entity.

instrumentation.subNet table

The subNet table contains a record of every discovered subnet address and mask.

Table 106. instrumentation.subNet database table schema

Column name	Constraints	Data type	Description
m_SubNet	• PRIMARY KEY • NOT NULL • UNIQUE	Text	The subnet address of a discovered subnet.
m_NetMask	• NOT NULL • UNIQUE	Text	The subnet mask of a discovered subnet.

instrumentation.vlan table

The vlan table contains a record of every discovered VLAN.

Table 107. instrumentation.vlan database table schema

Column name	Constraints	Data type	Description
m_Vlan	UNIQUE	Integer	The ID of the discovered VLAN.

instrumentation.frameRelay table

The frameRelay table contains a record of every discovered Frame Relay device.

Table 108. instrumentation.frameRelay database table schema

Column name	Constraints	Data type	Description
m_IfDlci	• PRIMARY KEY • NOT NULL • UNIQUE	Integer	The Frame Relay device Data Link Connection Identifier.
m_IfIndex	• PRIMARY KEY • NOT NULL	Integer	The unique value for each device interface.

instrumentation.ciscoFrameRelay table

The ciscoFrameRelay table contains a record of every discovered Cisco Frame Relay device.

Table 109. instrumentation.ciscoFrameRelay database table schema

Column name	Constraints	Data type	Description
m_UniqueKey	- NOT NULL - UNIQUE	Text	A combination of the IP Address, the FRIfIndex, and the FRDlci.
m_FRIfIndex	- PRIMARY KEY - NOT NULL	Integer	The unique value for each device interface.
m_FRDlci	- PRIMARY KEY - NOT NULL - UNIQUE	Integer	The Frame Relay device Data Link Connection Identifier.

instrumentation.hsrp table

The hsrp table contains a record of every discovered Hot Standby Router Protocol (HSRP) device.

Table 110. instrumentation.hsrp database table schema

Column name	Constraints	Data type	Description
m_GroupAddress	- PRIMARY KEY - NOT NULL - UNIQUE	Text	The group address of the device.
m_PrimaryAddress		Text	The primary address of the device.
m_StandbyAddress		Text	The standby address of the device.

instrumentation.pnniPeerGroup table

The pnniPeerGroup table contains the lowest level Peer Group Identifiers of PNNI devices that have been discovered. Logical PNNI Peer Groups IDs are not stored.

Table 111. instrumentation.pnniPeerGroup database table schema

Column name	Constraints	Data type	Description
m_PeerGroupId	- PRIMARY KEY - NOT NULL - UNIQUE	Text	The lowest level PNNI peer group identifier.

instrumentation.fddi table

The fddi table contains the Fibre Distributed Data Interface (FDDI) nodes that have been discovered.

Table 112. instrumentation.fddi database table schema

Column name	Constraints	Data type	Description
m_UniqueAddress	• PRIMARY KEY • NOT NULL	Text	The unique address of the node.
m_StationManagmentTask	• PRIMARY KEY • NOT NULL	Integer	The station management task for that node.

workingEntities database

The workingEntities database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table names are: workingEntities.finalEntity; workingEntities.containment.

The workingEntities database provides a central repository for information about discovered entities and the containment details associated with each of these entities. However, this database is populated only at the end of the discovery process.

workingEntities.finalEntity table

The finalEntity table is a central repository for information about discovered entities.

Table 113. workingEntities.finalEntity database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Unique name of the discovered entity.
m_Creator	NOT NULL	Text	Name of agent (or finder) that discovered the entity.
m_ObjectId		Text	Device class (a textual representation of the ASN.1 address).
m_Description		Text	Description of the device, taken from the sysDescr MIB variable for the entity.
m_UniqueAddress		Text	IP address of the network entity.
m_IsActive	Externally defined Boolean data type	Boolean Integer	Indicates whether the entity is active: (2) Indicates that the entity is discovered but is not in scope. Entities that are not in scope are not monitored by Network Manager. (1) Entity is active. (0) Entity is *in*active.

Table 113. workingEntities.finalEntity database table schema (continued)

Column name	Constraints	Data type	Description
m_HaveAccess	Externally defined Boolean data type	Boolean integer	Flag indicating whether SNMP access to the device is available: • 1: SNMP access is available • 0: No SNMP Access
m_EntityType	Externally defined entityType data type	Integer	Element type description of the discovered entity: • 0: Unknown type • 1: Base entity • 2: Local neighbor • 3: Remote neighbor
m_BaseName		Text	The name of the Base Entity for this device.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
m_ExtraInfo	Externally defined vblist data type	Object	Extra information requested by the agent.
m_LocalNbr	Externally defined vblist data type	Object	Information about the local neighbor.

workingEntities.containment table

The containment table is a central repository for information about containment information for discovered entities. It shows the containment relationships between all entities in the finalEntity table.

As an example of how the containment table works, assume the finalEntity table includes the following distinct entities:

- A device with IP address `1.2.3.4`
- An interface on this device, `1.2.3.4[0[1]]`

The finalEntity table provides no containment information for these two entities. In other words, it does not indicate that the interface 1.2.3.4[0[1]] is physically contained within the device 1.2.3.4. This containment information is held within the containment table, as follows:

```
m_Container='1.2.3.4'
m_Part='1.2.3.4[0[1]]'
m_IsPhysical=1
m_LinkType=1
```

Note that m_Container and m_Part are each unique names of entities on the network, each with a unique m_Name in the finalEntity table.

Table 114. workingEntities.containment database table schema

Column name	Constraints	Data type	Description
m_Container	• PRIMARY KEY • NOT NULL	Text	The name of an object which contains something. This object refers to an entity on the network and corresponds to an entity with its own entry and unique m_Name in the workingEntities.finalEntity table.
m_Part	• PRIMARY KEY • NOT NULL	Text	The name of the object which is contained. This object refers to an entity on the network and corresponds to an entity with its own entry and unique m_Name in the workingEntities.finalEntity table.
m_IsPhysical		Boolean integer	Flag indicating whether the containment is physical or logical: • 1: Physical Containment • 0: Logical Containment
m_LinkType		Integer	Value indicating mode of data transfer between `m_Container` and `m_Part`. The following values are possible: • 0: No data is transmitted. • 1: Data is transmitted both ways. • 2: Data travels from m_Container to m_Part. • 3: Data travels from m_Part to m_Container

workingEntities.interfaceMapping

The interfaceMapping table enables the stitching to quickly identify interfaces.

The following table lists the columns in the interfaceMapping table.

Note: Not all the fields in this table are populated; however, the use of this table provides a fast way of looking up data.

Table 115. workingEntities.interfaceMapping database table schema

Column name	Constraints	Data type	Description
m_Name	not null	Text	Unique name of an interface on the network.
m_IfIndex		Integer	SNMP ifIndex.
m_InterfaceId		Text	Interface identifier.
m_EntPhysIndex		Integer	Entity MIB physical Index if present.
m_IfDescr		Text	Interface RFC.ifDescr.
m_IfName		Text	Interface RFC ifName.
m_IfAlias		Text	Interface RFC ifAlias field.
m_IfType		Integer	Interface RFC ifType.
m_PhysAddress		Text	MAC address for this entity if present.

Table 115. workingEntities.interfaceMapping database table schema (continued)

Column name	Constraints	Data type	Description
m_BaseName	Not null	Text	Name of the "Base Entity" for this device.
m_AddressSpace		Text	Name of the address space this device is on. For public devices the field is null.

Working topology databases

The discovery engine, ncp_disco, uses a series of databases to perform the data processing stages of the discovery cycle. Stitchers operate on these databases to knit together a network topology and create the containment model.

The stitchers produce the various network topologies, such as layer 2 and layer 3 topologies, by amalgamating the information in the discovery agents returns tables into a single cumulative topology within the fullTopology database.

fullTopology database schema

The fullTopology database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table name is fullTopology.entityByNeighbor.

The fullTopology database holds the generated topology. On completion of the data collection phase of the discovery, the stitchers merge the information that has been retrieved by the discovery agents to form a single topology, which at this stage is in a name-to-name format.

fullTopology.entityByNeighbor table

The entityByNeighbor table contains information about connectivity between discovered devices.

Table 116. fullTopology.entityByNeighbor database table schema

Column name	Constraints	Data type	Description
m_Name	• PRIMARY KEY • NOT NULL	Text	Unique name of an entity on the network.
m_NbrName	• PRIMARY KEY • NOT NULL	Text	The name of the device that is connected to the unique network entity.
m_NbrType	Externally defined connectionType data type	Integer	Integer representation of the type of connection between the network entity and its neighbor: • 2: Main-to-Local • 3: Local-to-Remote

scratchTopology database schema

The scratchTopology database is defined in $NCHOME/etc/precision/ DiscoSchema.cfg. Its fully qualified database table name is: scratchTopology.entityByName.

The scratchTopology database holds the containment model that is derived from the fullTopology database (and created by stitchers). This is the version of the topology that is sent to the MODEL component.

Related concepts:

"Filters" on page 5
Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

Related tasks:

"Setting discovery filters" on page 28
Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

scratchTopology.entityByName table

The entityByName table contains the network model derived from the fullTopology database.

Table 117. scratchTopology.entityByName database table schema

Column name	Constraints	Data type	Description
`EntityName`	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Unique name of the network entity.
`BaseName`		Text	Unique base name of an entity.
`Address`		List of text	List of OSI model layer 1-7 addresses for the entity.
`Description`		Text	Value of sysDescr MIB variable or other suitable description of the entity.
`EntityType`	Externally defined entityTypes data type	Integer	Element type of the entity: • 0: Unknown • 1: Chassis • 2: Interface • 3: Logical interface • 4: Vlan object • 5: Card • 6: PSU • 7: Subnet • 8: Module

Table 117. scratchTopology.entityByName database table schema (continued)

Column name	Constraints	Data type	Description
EntityOID		Text	The device class to which the network entity belongs. This is a textual representation of the ASN.1 address.
Status	Externally defined Boolean data type	Boolean integer	Flag indicating the state of the entity: • 1: Active • 0: Not Active
IsActive	Externally defined Boolean data type	Boolean integer	Flag indicating whether or not the entity is active: • 1: Active • 0: Not Active
Contains		List of text	List of elements or other containers contained within the current network entity.
UpwardConnections		List of text	List of containers that contain this entity.
RelatedTo		List of text	List of entities that are connected to the network entity.
LingerTime		Integer	You can set the LingerTime for an entity in customized stitchers to determine how ncp_model handles the entity when ncp_disco sends the topology to ncp_model. The LingerTime value determines how many discoveries an entity can fail to be found in before it is assumed to have been removed from the network and its record is removed from the topology. If set to zero, the entity is deleted from ncp_model immediately when the discovery process updates the topology in ncp_model.
ExtraInfo	Externally defined vblist data type		Any additional information.

rediscoveryStore database

The rediscoveryStore database is used for comparison purposes in rediscovery mode. It is defined in $NCHOME/etc/precision/ DiscoSchema.cfg. Its fully qualified database table names are: rediscoveryStore.dataLibrary; rediscoveryStore.rediscoveredEntities

The rediscoveryStore database holds information from previous discovery cycles that can be used for comparison purposes during a full or partial rediscovery.

rediscoveryStore.dataLibrary table

The dataLibrary table is used as a reference point during rediscovery mode to compare the previous and present states.

Table 118. rediscoveryStore.dataLibrary database table schema

Column name	Constraints	Data type	Description
m_Name		Text	Unique name of an entity on the network.
m_UniqueAddress		Text	The IP address of a discovered network entity.
m_CompareDb	NOT NULL	Text	The entity that is used to compare this network entity.

rediscoveryStore.rediscoveredEntities table

The rediscoveredEntites table stores entities found during a rediscovery.

Table 119. rediscoveryStore.rediscoveredEntities database table schema

Column name	Constraints	Datatype	Description
m_Name		Text	Unique name of an entity on the network.
m_UniqueAddress		Text	The IP address of a discovered network entity.
m_PhysAddr		Text	The physical address of the entity.
m_OldBaseName			The base name of the entity prior to rediscovery
m_NewBaseName			The base name of the entity after rediscovery.

Topology manager database

The topology manager, ncp_model, stores the topology data following a discovery and sends the topology data to the topology database (NCIM), where it can be queried using SQL. When ncp_model starts up, it waits for the discovery engine to finish the discovery process, create the scratch topology, and insert it into the ncp_model database.

Table 120. MODEL (ncp_model) databases

Database	Description
master	The central store for the network topology.
model	Used to track topology updates.

master database schema

The master database is defined in $NCHOME/etc/precision /ModelSchema.cfg. Its fully qualified database table names are: master.entityByName; master.entityByNeighbor; master.containers. The master database holds all the network entities, their containment, and their connections.

master.entityByName table

The entityByName table holds information about all the discovered network entities. This table is active, populated with the information received from DISCO. Entries made into the entityByName table are also used to populate the containers table.

Table 121. master.entityByName database table schema

Column name	Constraints	Data type	Description
ObjectId	• PRIMARY KEY • NOT NULL • UNIQUE	Long integer	The unique Object ID of the network entity.
EntityName	• PRIMARY KEY • NOT NULL • UNIQUE	Text	Unique descriptive name of a network entity.
Address		List of text	List of OSI model layer 1 -7 addresses for the entity.
Description		Text	Value of sysDescr MIB variable or other suitable description of the entity.
EntityType	Externally defined entityTypes data type	Integer	Element type of the entity. • 0: Unknown • 1: Chassis • 2: Interface • 3: Logical interface • 4: VLAN object • 5: Card • 6: PSU • 7: Logical collection • 8: Module
ClassName		Text	Class name of network entity (if applicable).
EntityOID		Text	Value of sysOID MIB variable of the entity.
Status	Externally defined status data type	Integer	This field is populated by the Discovery engine, ncp_disco with the value of the field m_HaveAccess. This field therefore indicates whether ncp_disco acquired SNMP access to the device.
Security		Text	Password to access network entity (if applicable).
RelatedTo		List of text	List of entities that are connected to the network entity.

Table 121. master.entityByName database table schema (continued)

Column name	Constraints	Data type	Description
Contains		List of text	List of elements or other containers contained within the current network entity.
UpwardConnections		List of text	List of containers that contain this entity.
IsActive	Externally defined Boolean data type	Boolean Integer	Flag indicating whether an Active Object Class is needed: • 1: Active Object Class is needed • 0: Active Object Class is not needed
CreateTime		Time	Creation time of network entity record in table.
ChangeTime		Time	Time of last modification to the network entity record.
ActionType	Externally defined actions data type	Integer	The value of this field has significance when the record is broadcast on the message bus. It indicates the type of topology update that is being broadcast. This field can take the following values: **0** New **1** Update **2** Delete **3** Undefined
ExtraInfo	Externally defined vblist data type	Object	A list of extra information.
LingerTime	NOT NULL Default=3	Integer	The linger time is used during rediscovery so that the new topology can be merged with the existing topology. The value of LingerTime is decremented if the entity is not present in the new topology. The entity is only removed from the topology when the value of LingerTime reaches 0.

master.entityByNeighbor table

The entityByNeighbor table holds connectivity information for each network entity.

Table 122. master.entityByNeighbor database table schema

Column name	Constraints	Data type	Description
LeftId	• PRIMARY KEY • NOT NULL	Long integer	The Object ID of the left-hand side connection.
LeftName	• PRIMARY KEY • NOT NULL	Text	The entity name of the left-hand side connection.

Table 122. master.entityByNeighbor database table schema (continued)

Column name	Constraints	Data type	Description
`RightName`	• PRIMARY KEY • NOT NULL	Text	The entity name of the right-hand side connection.
`Speed`		Long64	The speed of the connection in bits per second (bps).
`Protocol`	Externally defined protocol data type	Integer	The transmission protocol type used by the connection.
`RelType`	Externally defined connectionType data type	Integer	The type of relationship.
`Duplex`	Externally defined Boolean data type	Boolean Integer	Flag indicating whether link is bidirectional (that is, full duplex): • 1: Link is bidirectional. • 0: Link is not bidirectional.

master.containers table

The containers table uses the containment model to consider each network entity as being contained by other network entities. The table, which is automatically populated as a result of entries made into the entityByName table, shows the parent of each entity, that is, the object that contains the present entity.

Table 123. master.containers database table schema

Column name	Constraints	Data type	Description
`ObjectId`	• PRIMARY KEY • NOT NULL	Long integer	The unique Object ID of the network entity.
`EntityName`	• PRIMARY KEY • NOT NULL	Text	Descriptive name of container network entity.
`MemberName`	NOT NULL	Text	The member name of the contained object.

model database schema

The model database is defined in $NCHOME/etc/precision/ ModelSchema.cfg. Its fully qualified database table names are: model.config; model.statistics. This database stores information about the topology so that during rediscovery, topologies can be merged efficiently.

model.config table

The model.config table stores the configuration information that is used by MODEL during rediscovery.

Table 124. model.config database table schema

Column name	Constraints	Data type	Description
LingerTime	• PRIMARY KEY • NOT NULL • UNIQUE	Integer	LingerTime value for the topology.
ChassisCreation Events	NOT NULL	Boolean Integer	Generates ItnmEntityCreation and ItnmEntityDeletion events for chassis entities.
IpInterfaceCreation Events	NOT NULL	Boolean Integer	Generates ItnmEntityCreation and ItnmEntityDeletion events for interfaces that have their own IP address.
MaintenanceState Events	NOT NULL	Boolean Integer	Generates ItnmMaintenanceState events for chassis entities and for interfaces that have their own IP address.
ManagedStatusUpdate Interval	NOT NULL	Integer	Interval in seconds at which ncp_model scans the NCIM managedStatus table for changes. This is the maximum time the poller should take to react to changes in managed status made in any of the following GUIs: Network Views, Network Hop View, Structure Browser. Default value 30 seconds.
DiscoveryUpdateMode	NOT NULL	Integer	For internal system use only. Prior to a batch update, **ncp_disco** sets this value to 1 for a partial discovery, or to 0 for a full discovery.

Any combination of the flags ChassisCreationEvents, IpInterfaceCreationEvents, and MaintenanceStateEvents can be turned on and off. The default is for all three to be disabled.

Note: If you have a network that contains routers with a large number of IP addresses, then enabling the IpInterfaceCreationEvents flag can might generate a large number of events in the Object Server.

model.profilingData

The model.profilingData table stores data associated with time and memory expended during the discovery. This table includes information on how long it took to transfer the discovery profiling data to the NCIM topology database.

Table 125. model.profilingData database table schema

Column name	Constraints	Data type	Description
BatchStartTime	• PRIMARY KEY • NOT NULL • UNIQUE	Integer	The time that a batch update from the Discovery engine, ncp_disco, started.
BatchStartSize	NOT NULL	Integer	Number of records in the batch received.
BatchStartMem	NOT NULL	64-bit integer	Memory usage when batch started.
BatchEndTime		Integer	The time a batch update from the Discovery engine, ncp_disco, ended.

Table 125. model.profilingData database table schema (continued)

Column name	Constraints	Data type	Description
BatchEndSize		Integer	Number of records at the end. **Note:** This value could be larger than at the start if subsequent batches got merged in.
BatchEndMem		64-bit integer	Memory usage when batch ended.
EntityCount		Integer	Number of entities after the batch update.
ChassisCount		Integer	Number of chassis devices after the batch update.
InterfaceCount		Integer	Number of interfaces after the batch update.

model.statistics table

The model.statistics table stores information about previous discoveries.

Table 126. model.statistics database table schema

Column name	Constraints	Data type	Description
TopologyCount	• PRIMARY KEY • NOT NULL • UNIQUE	Long	A count of the number of times the topology has been sent from DISCO to MODEL.
TopologySendFinished		Integer	Indicates whether DISCO has finished transferring the topology to MODEL. This column is set to 0 when the SendTopologyToModel.stch stitcher begins sending the topology, and set to 1 when it has completed sending the topology.
InsertCount		Long	The number of entities inserted into the topology.
UpdateCount		Long	The number of entities updated in the topology.
DeleteCount		Long	The number of entities deleted from the topology.

Failover database

Failover recovery with the failover database is not to be confused with agent and finder failover recovery, which are configured directly from the disco.config table. When selected, agent and finder failover recovery operate regardless of whether recovery with the failover database is implemented.

If the m_WriteTablesToCache column of the disco.config table is set to 1 (true), data is cached during the discovery process to enable data recovery in the event that the Discovery engine, ncp_disco, fails. A discovery running in this mode is slower than a standard discovery, because of the extra time required to store data on the disk throughout the discovery process.

Ignored cached data

If DISCO is restarted in failover recovery mode, any cached data for a group of tables are ignored.

The cached data for the following tables are ignored when DISCO is restarted in failover recovery mode:

- disco.config
- disco.managedProcesses
- disco.agents
- The entire scope database
- failover.config
- failover.doNotCache
- failover.restartPhaseAction

For the above tables, only the insertions specified in the schema file at the time of the restart are registered.

The failover database schema

The failover database is defined in $NCHOME/etc/precision/DiscoSchema.cfg. Its fully qualified database table names are: failover.config; failover.status; failover.findRateDetails; failover.doNotCache; failover.restartPhaseAction.

failover.config table

There must never be more than one insert into the failover.config table.

Table 127. failover.config database table schema

Column name	Constraints	Data type	Description
`m_InitialiseFromCache`	Externally defined Boolean data type	Boolean integer	Flag indicating whether to use the data that already exists in the cache: • 0: Do not use cached data • 1: Use cached data if any exists
`m_NumberOfRetries`		Integer	The number of times to try using the cached data before giving up (that is, the number of subsequent times that DISCO can be restarted before starting with a clean slate). If no value is specified, DISCO always starts with clear databases.
`m_StoreEveryNthDevice`	Default = 10	Integer	How often the findRateDetails table is to be updated. After the specified number of devices have been found the table is updated.

failover.status table

The failover.status table displays the number of times that the DISCO process has attempted to restart with cached data. This table is active, so you must not configure inserts into it.

Table 128. failover.status database table schema

Column name	Constraints	Data type	Description
m_NumberOfAttempts	• NOT NULL • PRIMARY KEY	Integer	The number of times that the DISCO process has attempted to restart with cached data. This column is set to 1 when DISCO is first run in failover recovery mode and incremented each time DISCO is subsequently run in failover mode.

failover.findRateDetails table

The findRateDetails table gives details of devices that have been found at a certain point in the discovery. This table is active and inserts must not be made in the schema file; the table is populated automatically.

Table 129. failover.findRateDetails database table schema

Column name	Constraints	Data type	Description
m_StartTime	• NOT NULL • PRIMARY KEY	Text	The time at which the first device was found.
m_LastFindTime		Text	The time at which the last device was found.
m_DevicesFound		Integer	The number of devices found so far.

failover.doNotCache table

To prevent caching a given table, you can specify its name in the doNotCache table. This ensures that unnecessary cache files are not created, such as those for temporary tables defined within stitchers.

Table 130. failover.doNotCache database table schema

Column name	Constraints	Data type	Description
m_DatabaseName	NOT NULL	Text	The name of any database that is not to be cached during failover recovery. The following tables must be cached in order to use the failover recovery mode, and therefore must not be listed in this table: • disco.status • failover.status The following tables must be cached, and therefore must not be listed in this table: • The agent despatch and returns tables. • finders.processing • translations.ipToBaseName
m_TableName	NOT NULL	Text	The name of the table within the database specified in m_DatabaseName that is not to be cached. Use * to indicate all the tables of the database.

failover.restartPhaseAction table

The restartPhaseAction table contains the set of stitchers that are executed when restarting in a given discovery phase. Multiple stitchers can be specified, but they are executed in an arbitrary order. It is recommended that at least the FinalPhase stitcher is executed when restarting in the topology creation phase.

Table 131. failover.restartPhaseAction database table schema

Column name	Constraints	Data type	Description
m_RestartPhase	NOT NULL	Integer	The phase in which DISCO is restarted.
m_ExecuteStitcher	NOT NULL	Text	The stitcher that is to be executed in this phase.

Example failover database configuration

This example uses OQL commands to insert configuration values into the failover database tables that are appended to the DiscoConfig.cfg file to configure DISCO when it is launched.

Example configuration of the failover.config table

This example uses OQL commands to insert configuration values into the failover.config table.

For this configuration of the failover.config table, data already in the cache is used. The Discovery engine, ncp_disco, can be restarted up to three times before cached data is ignored. These values are used only when disco.config.m_WriteTablesToCache=1.

```
insert into failover.config
(
        m_InitialiseFromCache,
        m_NumberOfRetries
)
values
(  1, 3  );
```

Example configuration of the failover.doNotCache table

This example uses OQL commands to insert configuration values into the failover.doNotCache table. The disco.config table and all tables of the instrumentation database are not cached.

```
insert into failover.doNotCache
(
        m_DatabaseName,
        m_TableName
)
values
(
        'disco', 'config'
);

insert into failover.doNotCache
(
        m_DatabaseName, m_TableName
)
values
(
        'instrumentation', '*'
);
```

Agent Template database

The databases of each discovery agent are based on a template called the agentTemplate database.

The agentTemplate database is defined in $NCHOME/etc/precision/DiscoSchema.cfg, and its fully qualified database table names are: agentTemplate.despatch and agentTemplate.returns.

Related reference:

"Discovery agent definition files" on page 50
The discovery agent definition files define the operation of the discovery agents.

Discovery agent despatch table

When a device has been interrogated by the Details agent, it is passed to the Associated Address agent to check whether it has already been discovered. If the device has not been discovered, the device details are processed and sent by a stitcher to the despatch table of the appropriate agent.

The despatch table is described in Table 132.

When the device details are placed in the despatch table, the agent attempts to retrieve connectivity information pertaining to the device.

Table 132. agentTemplate.despatch database table schema

Column name	Constraints	Data type	Description
m_Name	PRIMARY KEY NOT NULL	Text	Unique name of an entity on the network.
m_UniqueAddress	NOT NULL	Text	Unique IP address of the network entity.
m_ManagerId	PRIMARY KEY NOT NULL	Text	Manager of the device. If the device is accessed directly, this is set to " ". By default, this is set to " ".
m_Protocol		Integer	The protocol of the discovered device: • (1) IP • (2) IP-NAT
m_ObjectId		Text	Textual representation of the device class (an ASN.1 address).
m_SnmpAccessIP		Text	If present, overrides the IP address used for SNMP access to devices using the Helper Server.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
m_HaveAccess	Externally defined Boolean data type	Boolean Integer	Flag indicating whether there is SNMP access to the device: • (1) Have Access • (0) No Access

Discovery agent returns table

Returned device connectivity details are placed in the returns table of the agent. These details are used to populate the topology databases.

The returns table is described in Table 133.

Table 133. agentTemplate.returns database table schema

Column name	Constraints	Data type	Description
m_Name	NOT NULL	Text	Unique name of an entity on the network.
m_UniqueAddress	NOT NULL	Text	Layer 3 address of this entity.
m_Protocol		Integer	The protocol of the discovered device: • (1) IP • (2) IP-NAT
m_ObjectId		Text	Textual representation of the device class (an ASN.1 address).
m_HaveAccess	Externally defined Boolean data type	Boolean Integer	Flag indicating whether there is SNMP access to the device: • (1) Have Access • (0) No Access
m_ExtraInfo	Externally defined vblist data type	Object	Any extra information specified by the user in the agent definition file.
m_LocalNbr	Externally defined neighbor data type	Object	Direct neighbors (interfaces).
m_RemoteNbr	Externally defined nbrsNeighbor data type	Object	Remote neighbors connected to interfaces.
m_UpdAgent		Text	The agent that updated this device.
m_SnmpAccessIP		Text	If present, overrides the IP address used for SNMP access to devices using the Helper Server.
m_AddressSpace		Text	The name of the NAT address space to which the device belongs. This value is set in the translations.NATAddressSpaceIds table. If the discovery is not using NAT, or if the device is in the public domain, this value is NULL.
m_LastRecord	Externally defined Boolean data type	Boolean integer	Is this the last record for this entity: • (1) True • (0) False

Appendix B. Discovery process

The Network Manager discovery process produces a network topology that includes connectivity and containment data.

Discovery subprocesses

The discovery process consists of several subprocesses that work together to discover devices and device interconnectivity.

When you launch a discovery, the internal Network Manager discovery engine (`ncp_disco`) is run. The ncp_disco engine manages the process of discovering device existence and interconnectivity.

Whenever you launch a full discovery the Discovery Engine, ncp_disco, rereads its configuration files. The Discovery Engine also instructs the Helper Server and the individual helpers to reread their configuration files. This is controlled by the `DiscoReadConfig()` rule within the FullDiscovery stitcher file.

Note: When you launch a partial discovery, ncp_disco does not read its configuration files.

The discovery engine operates by detecting the existence of a device on the network and querying the device for inventory and connectivity information, which is subsequently processed or 'stitched' together to generate a connectivity or topology model. The discovery engine components are described in Table 134.

Table 134. Discovery components

Name	Description
Finders	Finders discover the existence of devices but do not retrieve connectivity information.
Agents	ncp_disco uses discovery agents to request connectivity information from devices that the finders have discovered. There are a variety of agents, each specialized to retrieve information from different devices, and, in certain cases, to use different protocols. Agents do not have any direct interaction with the network, but instead retrieve information through the Helper Server. Agents can be libraries or text files, and are specialized for particular protocols, devices or classes.
Helper Server	The Helper Server manages the helpers and stores the information that is retrieved from the network. Discovery agents retrieve their information through the Helper Server to reduce the load on the network. The Helper Server can service the requests directly with cached data or pass on the request to the appropriate helper.
Helpers	The helpers retrieve information from the network on behalf of the discovery agents. Helpers also translate agent queries into the appropriate network protocol and make requests to the devices.

Table 134. Discovery components (continued)

Name	Description
Stitchers	Stitchers are processes that transfer, manipulate and distribute data between databases. The discovery stitchers are also responsible for processing the information collected by the agents and using this information to create the network topology. A predefined set of stitchers is included with Network Manager. You can modify existing stitchers or write new stitchers to perform custom manipulation of your network topology. For example, you can write a stitcher to make your device interfaces appear with a custom naming convention. Stitchers are coded using the stitcher language.

Discovery timing

Each full discovery consists of one or more discovery cycles. The division of a full discovery into multiple discovery cycles enables the discovery to complete in a timely way.

In the first discovery cycle, Network Manager discovers the existence of a predetermined majority of devices on the network, and proceeds to complete all data collection and processing operations associated with these devices. When Network Manager has discovered the existence of a predetermined majority of devices on the network, Network Manager enters the *blackout state*.

Any devices that Network Manager discovers during the blackout state are placed into a database table named finders.pending. These devices are only processed in the following discovery cycle. This means that the discovery process does not have to wait for all devices to be discovered before proceeding to the more detailed data collection and data processing operations.

Note: Ideally a discovery should complete in a single discovery cycle; however, sometimes it is not possible to discover the existence of entities sufficiently quickly as a result more discovery cycles are needed. Reasons why the system does not discover the existence of entities sufficiently quickly include: ping sweeping of sparsely populated subnets, and lack of access to devices. First-time discoveries often have multiple cycles. This can be mitigated by using the BuildSeedList.pl script to build a seed list after the initial discoveries. This seed list will then be used in subsequent discoveries to find devices in a more timely manner.

By default, each discovery cycle is made up of a data collection stage and a data processing stage. The data collection stage is in turn broken up into three phases. Figure 1 on page 279 shows a timing diagram for a discovery that requires two discovery cycles to complete.

The data collection and data processing stages are briefly described in Table 135 on page 279.

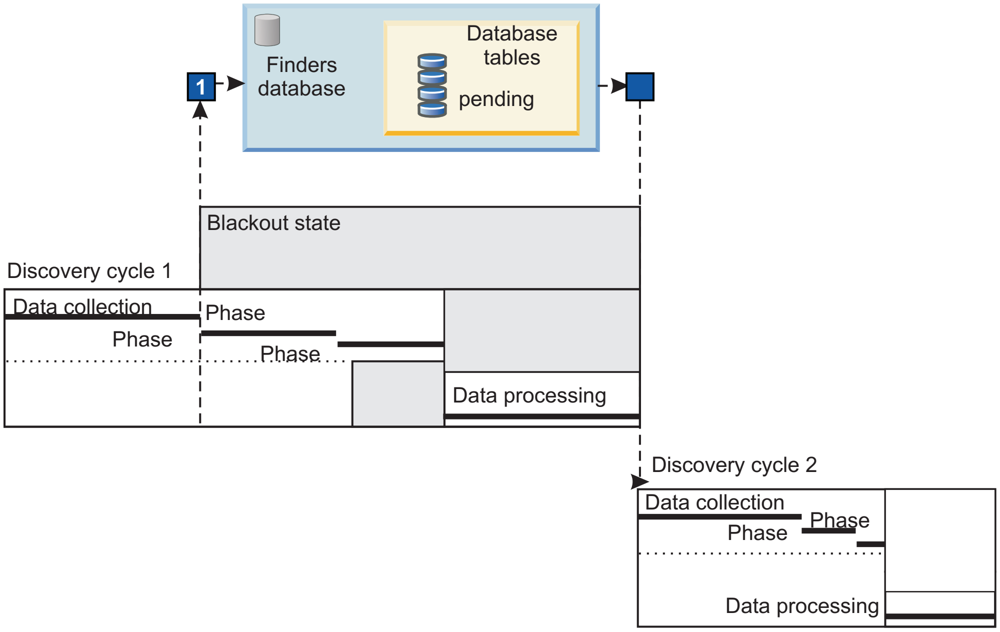

Figure 1. Discovery timing for a full discovery with two discovery cycles

In Figure 1, the blackout state for the first discovery cycle begins and ends at the instants indicated by the numbers 1 and 2 respectively:

1 : *Blackout state begins*. A predetermined majority of devices on the network have now been discovered. Any devices discovered after this point are placed into the finders.pending table for processing in the subsequent discovery cycle.

2 : *Blackout state ends*. Devices stored in the finders.pending table are now processed in the subsequent discovery cycle.

Note: If the network being discovered is particularly large or complex, more than two discovery cycles may be required to complete a full discovery. In this case, each discovery cycle, except for the last cycle, has its own blackout state.

Table 135. Data collection and data processing stages

Stage or Phase	Description
Data collection stage	During this stage, Network Manager interrogates the network for device information, using the finder, agent and helper components of DISCO. The data collection stage is divided into three phases, which are described in this table.
Data collection: first phase	During this phase, finders identify devices on the network. Phase one completes when the device *find rate* drops below a certain level. For each device discovered, agents retrieve device details, IP addresses associated with the device, and device connectivity information.
Data collection: second phase	During this phase, an agent retrieves IP address to MAC address mapping data.
Data collection: third phase	During this phase agents download all forward database table information for the network switches and ping all devices to confirm the accuracy of the contents of the forward database tables.

Table 135. Data collection and data processing stages (continued)

Stage or Phase	Description
Data processing stage	During this stage, Network Manager deduces the network topology based on data collected during the data collection stage. Stitchers analyze the data collected and build a network topology that includes connectivity and containment data.

Related concepts:

"Discovery stages and phases"
The discovery process can be divided into two stages: data collection and data processing. The stages are subdivided into phases.

"Discovery cycles" on page 285
A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

"Data collection stage" on page 281
The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

"Data processing stage" on page 281
Topology deduction takes place during the data processing stage, as the information from the data collection stage is analyzed, interpreted and processed by the stitchers. The culmination of the data processing stage is the production of the containment model.

Discovery stages and phases

The discovery process can be divided into two stages: data collection and data processing. The stages are subdivided into phases.

Related concepts:

"Discovery timing" on page 278
Each full discovery consists of one or more discovery cycles. The division of a full discovery into multiple discovery cycles enables the discovery to complete in a timely way.

"Discovery cycles" on page 285
A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

Related tasks:

"Monitoring discovery progress" on page 131
You can use the **Monitoring** pane to monitor the progress of the current discovery through each of the discovery phases.

Data processing stage

Topology deduction takes place during the data processing stage, as the information from the data collection stage is analyzed, interpreted and processed by the stitchers. The culmination of the data processing stage is the production of the containment model.

The data processing stage corresponds to creating the topology. This is the final conceptual step in the discovery cycle.

The data processing and data collection stages usually overlap, because you can configure the stitchers to begin processing connectivity information from different discovery agents before the main stitching operation begins.

Related concepts:

"Discovery timing" on page 278
Each full discovery consists of one or more discovery cycles. The division of a full discovery into multiple discovery cycles enables the discovery to complete in a timely way.

"Creating the topology" on page 292
The creation of the topology is carried out in several steps.

Data collection stage

The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

First phase

In the first phase of data collection, the finders identify all the devices that exist on the network. Generally, a phase can be completed when all the launched processes have completed their operation. However, although you might want to wait until all devices have been discovered by the finders before proceeding to phase two, it is inefficient to hold back the discovery process by waiting indefinitely. The first phase therefore completes when the *find rate* drops below a certain level, determined by no devices being discovered for the amount of time specified in `disco.config.m_NothingFndPeriod`.

The following conceptual steps in the discovery cycle take place during data collection phase one:

- Discovering device existence
- Discovering device details (standard)
- Discovering associated device addresses
- Discovering device connectivity

Agents in the first phase

Some agents return data that can be used to find other devices, for example, the IP address of remote neighbors, or the subnet within which a local neighbor exists. This mechanism is known as *feedback*.

The Feedback stitcher manages feedback by sending the information returned by the agents to the Ping finder for inclusion in the discovery. However, the blackout state ensures that any agent involved in the feedback process must be run in phase one for devices to be discovered in the current discovery cycle.

Phase one also usually involves the Switch discovery agents downloading all VLAN and interface information.

Blackout state

After phase one, the discovery enters the *blackout state*. The finders have discovered the existence of a pre-determined majority of devices on the network. Any new device addresses discovered in the blackout state, either by the finders or recursively by a discovery agent, are put into the finders.pending database table.

Devices in the finders.pending database table are processed in the next discovery. If there are devices in the finders.pending database table, the next discovery starts as soon as the current discovery finishes.

Second phase

After the criteria for the completion of phase one have been fulfilled, phase two begins. To map layers two and three of the OSI model, the ARP Cache discovery agent populates the Helper Server with ARP data, which is a list of device IP address-to-MAC address resolution.

Before the discovery can transfer from phase two to phase three, the processes from phase two must have completed their operation. An agent is considered to have finished after all entities in its despatch table are also in its returns table.

The agents are multithread, and records of discovered devices passed to the agents are tagged with a certain phase. Consequently, at any time an agent can be processing devices in two separate phases. If any action that should have occurred in phase two is detected after phase three has begun, phase three continues while the agent runs through phase two processing.

Third phase

By phase three, the discovery process has full knowledge of the devices that exist within the network (acquired from phase one) and access to full IP address-to-MAC address mappings for all devices in the Helper Server (acquired from phase two). The Switch agents can now proceed to download all the forward database table information of the network switches whilst pinging all devices to confirm the accuracy of the contents of the forward database tables.

When phase three has finished, which is signified by the completion of all processes scheduled to run in the phase, the discovery is ready to proceed from the data collection stage to the data processing stage, where all the connectivity information is knitted together to form a network topology.

Impact of the stages and phases approach on DISCO processes

The division of the data collection stage into phases affects all the processes involved in the discovery and network topology deduction, because the phases are processed in order. Any given phase cannot begin until the criteria for completion of the previous phase have been met.

All the processes of DISCO must therefore have an associated phase (or phases) in which they are allowed to operate. Thus, whilst the finders are typically configured to run through all phases, you might want to configure certain discovery agents to operate only within a specific phase(s). The flexibility of DISCO allows you to have processes that are intelligent enough to behave differently when they operate within different phases, and can pass control to other processes or stop operation until the start of their next operational phase.

Related concepts:

"Discovery timing" on page 278
Each full discovery consists of one or more discovery cycles. The division of a full discovery into multiple discovery cycles enables the discovery to complete in a timely way.

"Discovering device existence" on page 286
The discovery of device existence is carried out in several steps.

"Discovering device details (standard)" on page 287
The standard discovery of device details is carried out in several steps.

"Discovering associated device addresses" on page 289
There are several steps in the process flow during the discovery of associated device addresses.

"Discovering device connectivity" on page 291
The discovery of device connectivity is carried out in several steps.

Advantages of staged discovery

There are several reasons why it is advantageous to apply a staged and phased approach to discovery.

Switch connectivity

In determining the connectivity of some devices, it is sometimes necessary for the discovery agent to know all the devices that exist before requesting particular Management Information Base (MIB) variable(s), especially if the requested information is transient.

An example is when the layer 2 agents discover connectivity between Ethernet switches. Ethernet switches have forward database tables that expire over time. So, to ensure that a switch has a fully populated forward database table at the time of interrogation, you could ping all devices associated with the switch.

You would therefore configure the switch discovery agents to perform some other processing in data collection phase one. After the agents receive the signal that phase one has been completed (that is, all devices have been found) they can start phase two operations. For example, they could ping all devices within the discovery domain while downloading the forward database tables for all switches.

Mapping subnet boundaries

One limitation of configuring individual discovery agents to make individual ARP requests directly from the Helper Server is that the ARP helper cannot run simultaneously on multiple subnets unless it is specifically configured to do so. To resolve this problem, use a special ARP Cache discovery agent that imitates a generic discovery agent (in the sense that entities can be sent to it) but that also can map boundaries or different layers of the OSI model.

The ARP Cache discovery agent can inquire about ARP caches that exist on routers. It uses this information to populate the ARP helper database within the Helper Server and build up full device IP address to MAC address mapping without having to rely on the ARP helper.

This approach can be applied when using switch discovery agents that need to perform IP address-to-MAC address resolution before they can start operation. Following the example above, you could configure your discovery data collection stage to have three phases:

- Phase one: Find all devices that exist on the network.
- Phase two: Use the ARP Cache discovery agent to populate the Helper Server with full IP address to MAC address mappings.
- Phase three: Ping all devices and invoke the switch discovery agents by downloading the forward database tables for all switches in the network, using the IP address to MAC address mappings determined in phase two.

Multiphase discovery agents

Another possible consequence of dividing the data collection stage into phases is that you can configure the discovery agents to perform different operations within different phases.

Although a discovery agent is programmed to start operating in phase two, it could also conduct some other operation in phase one. This is because the end of phase one signifies only that all devices have been discovered. The agent could be configured to perform other actions such as downloading interfaces, issuing Telnet requests, or downloading other MIB variables during phase one. Only after phase two has started does the agent begin to process instructions specific to phase two.

Tip: It is good practice to configure the discovery to occur over multiple phases, to ensure maximum accuracy of the deduced topology.

Effect of discovery multiphasing on network traffic

One of the main benefits of multiphasing is reduced network traffic.

Because similar types of network requests are grouped in phases, data can be cached in the Helper Server to reduce the network load. The Helper Server is the intermediary between the discovery agents and the network, and can amalgamate multiple pings of the same device into one block so that they are resolved into a single ping.

The Helper Server also has a request pool that ensures that the Helper Server does not overload the network. The request pool does this by restricting the number of simultaneously-handled requests.

Criteria for multiphasing

The main criterion for configuring a discovery that has multiple phases is to assess the requirements of the different operations that need to be performed during the discovery process. For example, Ethernet-based discovery agents require at least two phases. It is possible to have discovery agents that can operate in any phase.

Managing the phases

The different phases of the discovery data collection stage are managed by an internal *phase manager*.

The phase manager:

- Reads the maximum overall phase number and calculates the total number of phases when all the discovery agent and stitcher definition files are loaded.
- Calculates the phase and process dependencies, that is, which discovery agents are scheduled to run in which phases.
- Monitors the processes running during the phases.

When the phase manager detects that all the processes for the current phase have completed, it sends a signal indicating phase completion for all the processes that are waiting to be launched in the next phase.

Discovery cycles

A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

The discovery data flow can be categorized into the following conceptual steps:

- Discovering device existence
- Discovering device details (standard)
- Discovering device details (context-sensitive)
- Discovering associated device addresses
- Discovering device connectivity
- Creating the topology

These steps follow the discovery data flow in order from start to finish, with the exception of discovering device details (context-sensitive), which replaces discovering device details (standard) if the discovery is context-sensitive.

Related concepts:

"Discovery timing" on page 278
Each full discovery consists of one or more discovery cycles. The division of a full discovery into multiple discovery cycles enables the discovery to complete in a timely way.

"Discovery stages and phases" on page 280
The discovery process can be divided into two stages: data collection and data processing. The stages are subdivided into phases.

"Discovery process with EMS integration" on page 295
Network Manager collects topology data from an EMS using collectors.

Discovering device existence

The discovery of device existence is carried out in several steps.

Figure 2 shows how the initial existence of devices on the network is discovered.

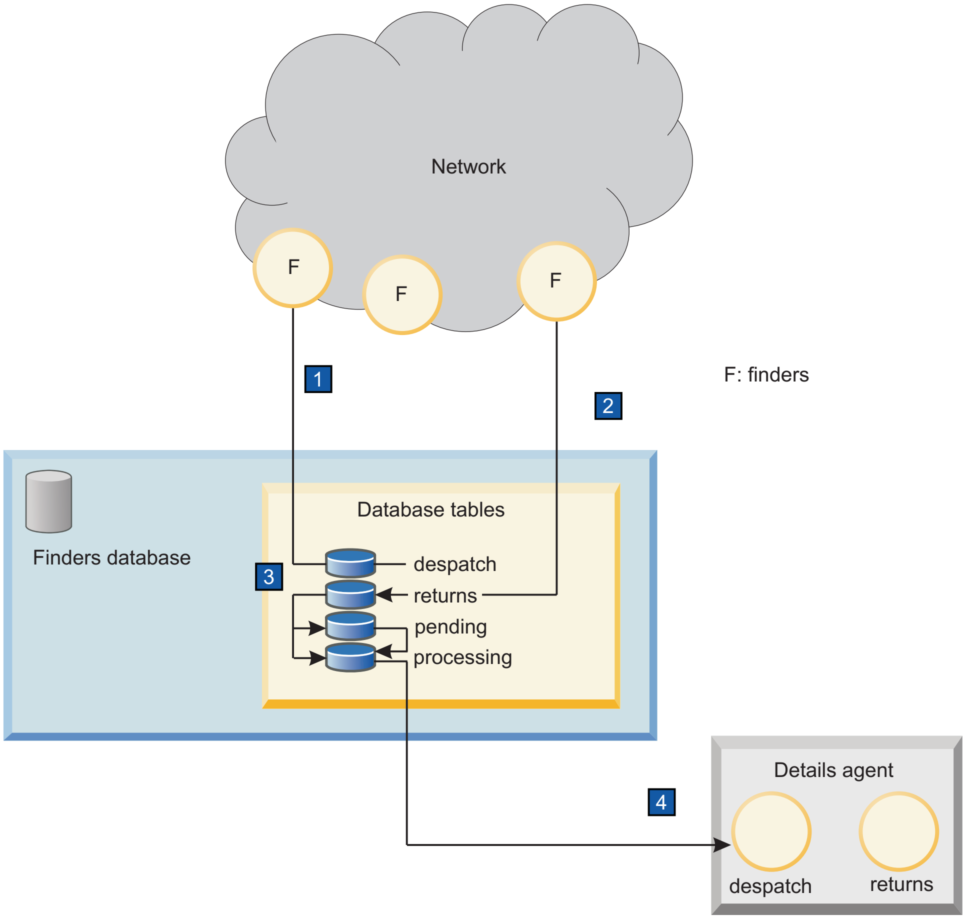

Figure 2. Discovery process flow: device existence

The process flow shown in Figure 2 is described below.

1 : The finders receive their instructions from their configuration files and the inserts made into the finders.despatch table, then proceed to the network to look for devices.

2 : The finders return the device existence information to the finders.returns table.

3 : After the device existence information is placed into the finders.returns table, a stitcher moves the information to the finders.processing table. This signifies that the network entity is being processed by DISCO. If the discovery is in the blackout state, the information is placed into the finders.pending table instead.

4 : A stitcher moves the information about device existence from the finders.processing table to the Details.despatch table, ready for processing by the Details agent.

Related concepts:

"Data collection stage" on page 281
The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

Discovering device details (standard)

The standard discovery of device details is carried out in several steps.

Figure 3 shows how device details are discovered in a standard discovery.

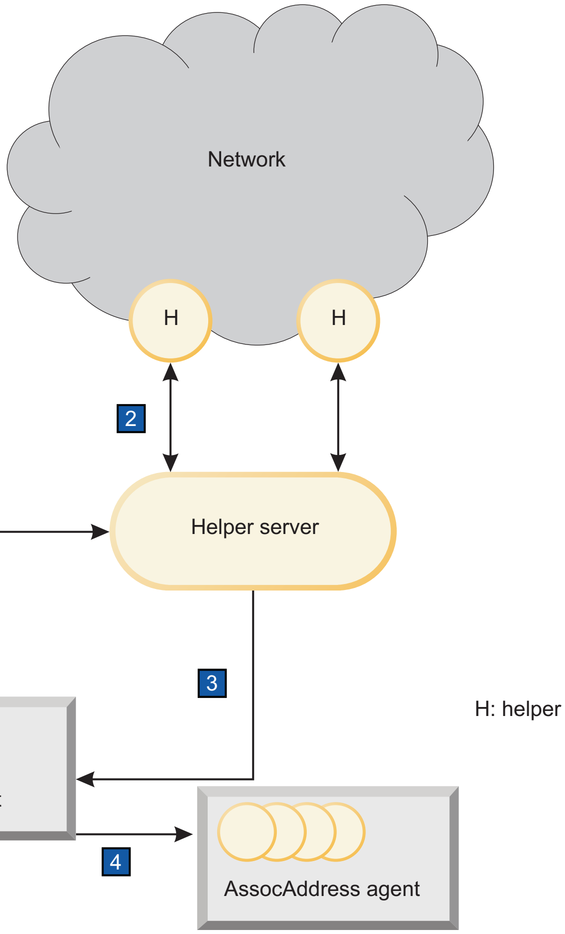

Figure 3. Discovery Process Flow: Device Details (Standard)

The process flow shown in Figure 3 is described below.

1 : All the agent despatch tables are active, so an insertion into the Details.despatch table automatically triggers the Details agent to discover basic device information and determine whether SNMP access to the device is available.

2 : The Details agent interrogates the network through the Helper Server. Requests are cached to reduce the number of times that the helpers (represented by the letter H in Figure 3) must interrogate the network directly.

3 : The information retrieved from the network is returned to the Details.returns table.

4 : The information in the Details.returns table is passed to the despatch table of the Associated Address (AssocAddress) agent for processing.

Related concepts:

"Data collection stage" on page 281
The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

Discovering device details (context-sensitive)

The discovery of context-sensitive device details is carried out in several steps.

Figure 4 shows how device details are discovered in a context-sensitive discovery.

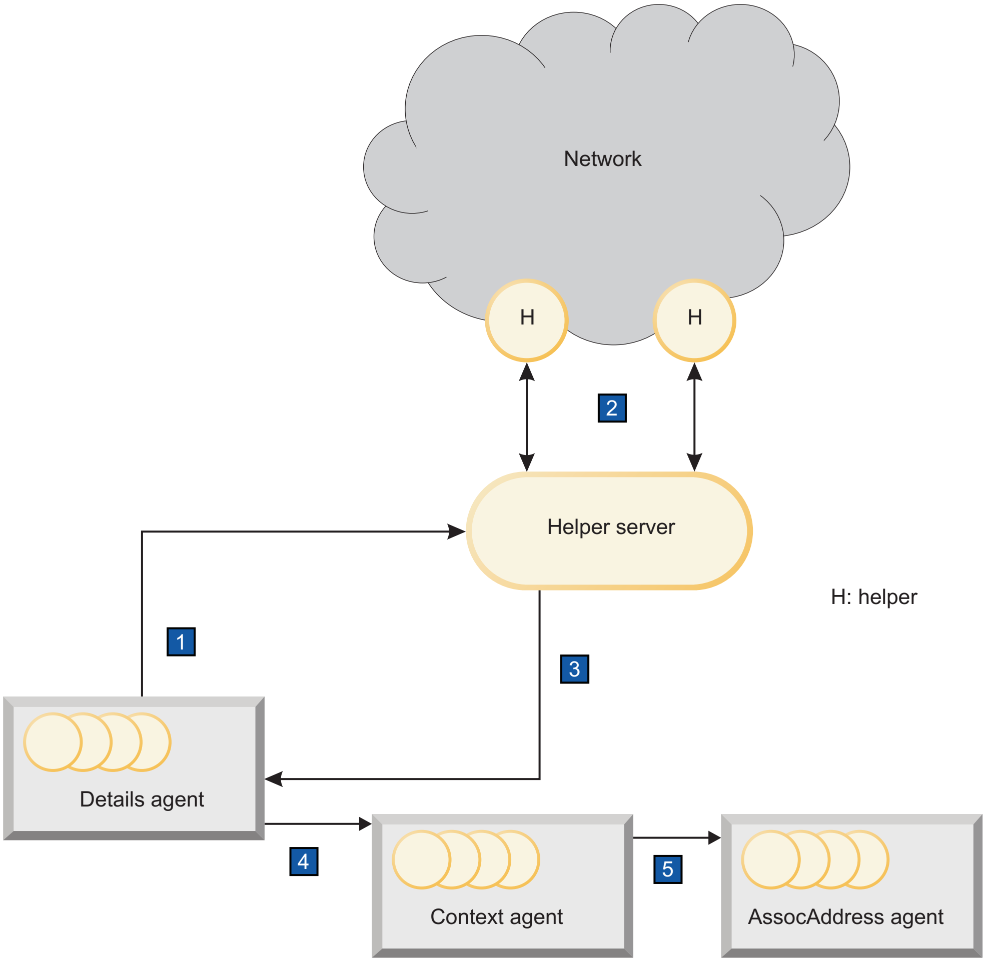

Figure 4. Discovery process flow: device details (context-sensitive)

The process flow shown in Figure 4 is described below.

1 : All the agent despatch tables are active, so an insertion into the Details.despatch table automatically triggers the Details agent to discover basic device information and determine whether or not SNMP access to the device is available.

2 : The Details agent interrogates the network through the Helper Server. Requests are cached to reduce the number of times that the helpers must interrogate the network directly.

3 : The information retrieved from the network is returned to the Details.returns table.

4 : The information in the Details.returns table is passed to the despatch table of the appropriate Context agent, which adds context tags.

5 : After the Context agent has finished its processing, the information is passed to the despatch table of the Associated Address (AssocAddress) agent for processing.

Related concepts:

"Context-sensitive discovery" on page 8
If you need to discover devices such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

Related tasks:

"Configuring a context-sensitive discovery" on page 102
If you have devices that you need to discover such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

Related reference:

"DiscoConfig.cfg configuration file" on page 63
The DiscoConfig.cfg configuration file is used to have the Ping finder automatically check the devices discovered by the File finder, and to enable a context-sensitive discovery.

Discovering associated device addresses

There are several steps in the process flow during the discovery of associated device addresses.

The following figure shows how associated device addresses are discovered.

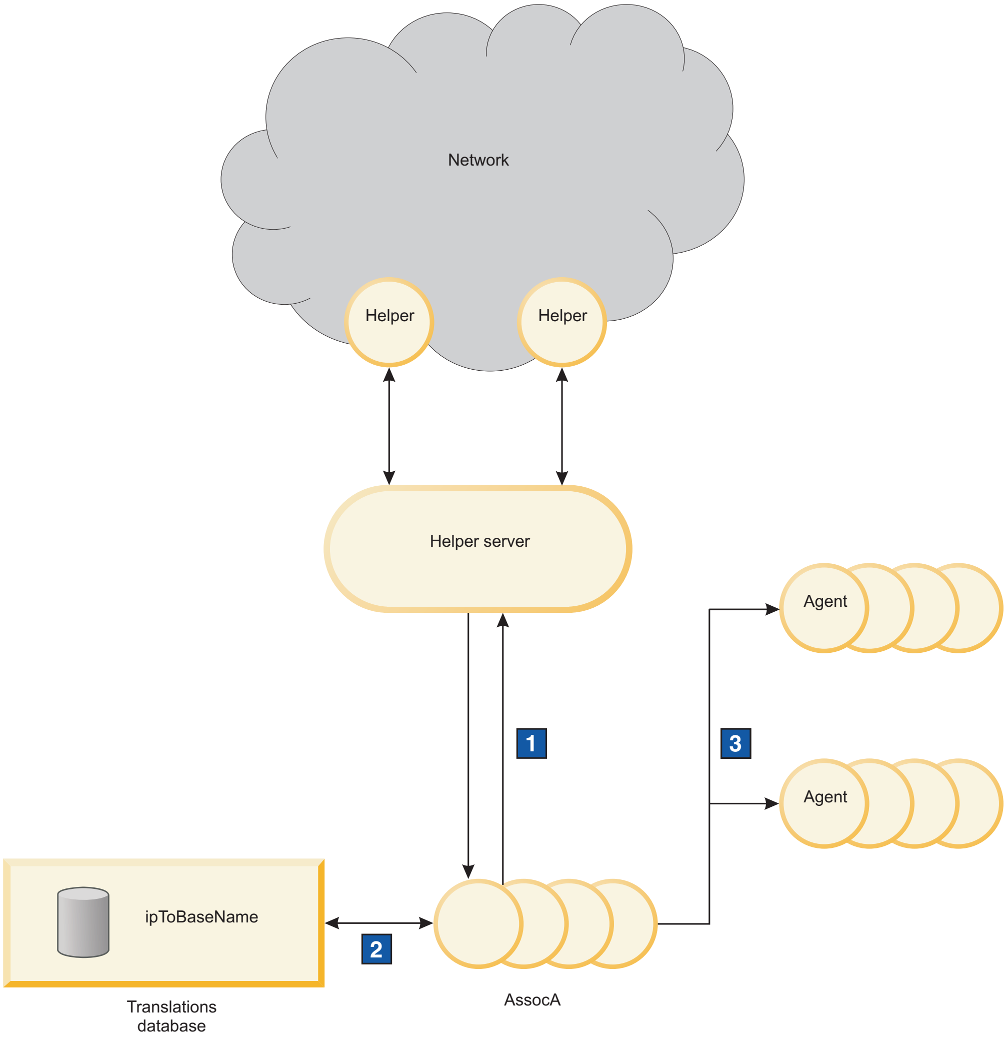

Figure 5. Discovery process flow: associated device addresses

The following process flow describes Figure 5:

1 : The Associated Address agent uses the Helper Server to download all the IP addresses associated with the interfaces of the device that is under investigation.

2 : The Associated Address agent checks the IP addresses against the registry of addresses, the translations.ipToBaseName table. The details are also added to this registry. If the device has already been discovered by another of its addresses (that is, if the translations.ipToBaseName table already contains a record for this device), the details of the device are not sent to the discovery agents.

3 : Provided the device has not already been discovered, the stitchers pass the details to the appropriate discovery agents, as specified in the `DiscoAgents.cfg` configuration file.

Related concepts:

"Data collection stage" on page 281
The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

Discovering device connectivity

The discovery of device connectivity is carried out in several steps.

The following figure shows how device connectivity is discovered, as well as how devices are discovered recursively.

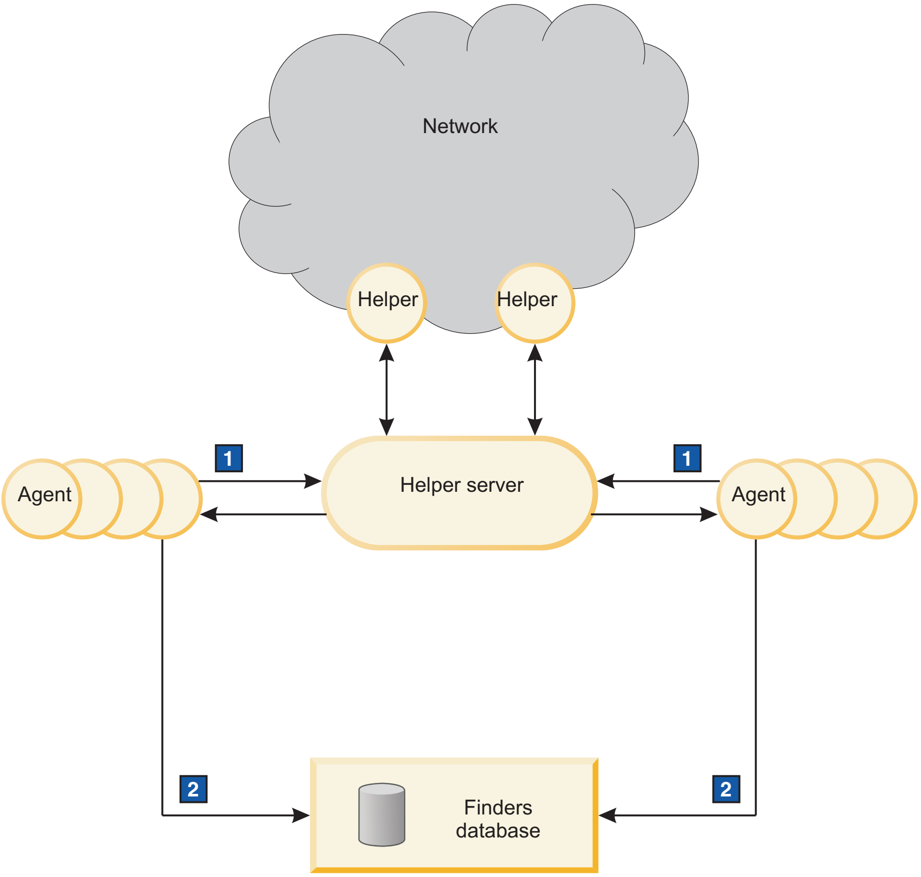

Figure 6. Discovery process flow: device connectivity

The following process flow describes Figure 6:

1 : When information is inserted into the `despatch` table of a discovery agent, the agent attempts to discover the connectivity information for that device. The agent sets up a TCP socket-based communication link with the Helper Server and requests the appropriate connectivity information.

2 : A stitcher passes the addresses of the remote neighbors of the device, and the subnet address or addresses of the device, to a finder for discovery. Because these addresses might not exist, and also might not be in the specified discovery scope, the addresses must run through the discovery process from the beginning.

Related concepts:

"Data collection stage" on page 281
The data collection stage involves interrogating the network for device information to produce a network topology. DISCO uses the finders, agents and helpers during the data collection stage. The data collection stage can be further subdivided into a number of phases.

Creating the topology

The creation of the topology is carried out in several steps.

The following figure shows a simplified data flow for the creation of the topology from the raw data returned by the discovery agents

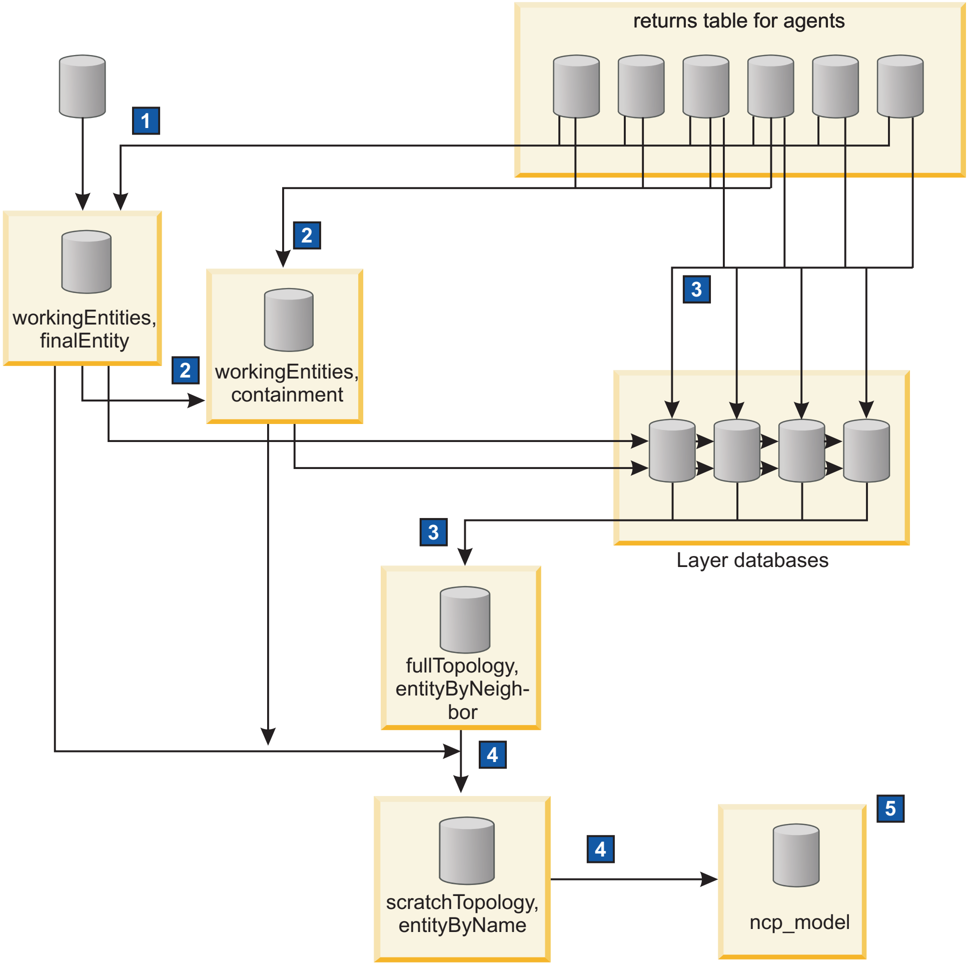
returns table for agents
1
2
3
workingEntities, finalEntity
2
workingEntities, containment
Layer databases
3
fullTopology, entityByNeigh-bor
4
4
5
scratchTopology, entityByName
ncp_model

Figure 7. Discovery process flow: creating the topology

The following process flow describes the data flow.

1 : After all the discovery agents have finished, and the discovery enters the data processing stage, special data processing stitchers interact with the discovery agent databases to produce the workingEntities.finalEntity table.

2 : The stitchers use a subset of the agents-returns tables, together with the workingEntities.finalEntity table, to deduce and create the containment model. This model is stored in the workingEntities.containment table.

3 : The stitchers use a further subset of the agents-returns tables, together with the workingEntities.finalEntity table and the workingEntities.containment table, to build the various topology layers, which are stored in the layer database tables. The full set of layers is merged in the fullTopology.entityByNeighbor table.

4 : The stitchers merge the three tables produced (workingEntities.finalEntity; workingEntities.containment; fullTopology.entityByNeighbor) to build the network model.

5 : The Topology manager, `ncp_model`, instantiates each network element (subject to the instantiation filter) and sends the topology to other components as required.

Related concepts:

"Data processing stage" on page 281
Topology deduction takes place during the data processing stage, as the information from the data collection stage is analyzed, interpreted and processed by the stitchers. The culmination of the data processing stage is the production of the containment model.

Broadcast of discovery data

On completion of a discovery the Topology manager, ncp_model, uses the message bus to receive topology updates from the discovery and to pass on those updates to the message bus where other processes like the Event Gateway can retrieve these updates. In addition, ncp_model also uses these updates to update the NCIM topology database.

Data is stored in two formats:

- NCIM cache format
- Legacy master.entityByName format

The purpose of each storage format is as follows:

NCIM cache format
Data in this format is used by ncp_model to place updates on the message bus for other processes such as the Event Gateway, ncp_g_event

Legacy master.entityByName format
Data in this format is used by ncp_model to update the NCIM topology database

The NCIM cache format is described in the *IBM Tivoli Network Manager IP Edition Topology Database Reference*.

Advanced discovery configuration options

Use this information to understand how to configure the discovery process data flow and to configure download of full routing tables.

Configurable discovery data flow

The discovery process data flow is user-configurable. Stitchers control the movement of data between databases, and you can customize the discovery process by changing the way in which the stitchers are triggered and operate.

Stitcher and agent triggers

You can modify the data flow by changing the criteria that trigger the deployment of the stitchers and discovery agents, by modifying the stitchers, and, if necessary, by modifying the agent definitions. Some typical triggers are:

- Data being inserted into a specific database table
- A stitcher or discovery agent completing its operation
- The end of a discovery phase

Any changes you make are automatically detected by DISCO during its periodic scan of the agent and stitcher files (the scan frequency is determined by the entry in the disco.config database). On detecting changes, DISCO modifies its agent and stitcher definitions databases accordingly, and applies the changes to the next discovery cycle.

For more details about the stitchers and the stitcher language, see the *IBM Tivoli Network Manager IP Edition Language Reference Guide*.

On-demand stitchers

Stitchers can be started on demand. If you insert a stitcher into the stitchers.actions database, DISCO automatically runs the stitcher. This means that the discovery cycle can be started at any point, and further actions can be configured to start when the stitcher completes.

Related reference:

"stitchers.actions table" on page 220
If a stitcher is inserted into the stitchers.actions table, DISCO runs the stitcher. Once the stitcher has completed, its entry is deleted from the stitchers.actions table. Any stitchers triggered to execute from the stitcher that has been inserted, or upon completion of the stitcher, are also executed.

Partial matching

By default, the discovery process uses partial matching, which means that the discovery agents do not need to download the full routing tables during discovery.

You do not need to modify the discovery agent definition files to use partial matching. However, it is possible to prevent the IpForwardingTable and IpRoutingTable discovery agents from using partial matching in certain cases if you have devices on your network that do not support partial matching.

To prevent partial matching on certain devices, you must specify the devices that do not support partial matching in the `DiscoRouterPartialMatchRestrictions();` section of the `IpForwardingTable.agnt` definition file (for modern devices that use RFC2096) or the `IpRoutingTable.agnt` definition file (for older devices that use RFC1213). If a discovered device matches the filter specified in the `DiscoRouterPartialMatchRestrictions();` section, partial matching is not attempted on that device.

Discovery process with EMS integration

Network Manager collects topology data from an EMS using collectors.

The following steps show how Network Manager collects topology data from an EMS using collectors.

Collector-based discovery can be divided into the following conceptual steps:

- Discovering device existence
- Discovering basic device information
- Discovering detailed device information

For an overview of how Network Manager collects topology data from Element Management Systems (EMSs) and integrates this data into the discovered topology, see the *IBM Tivoli Network Manager IP Edition Product Overview*.

Related concepts:

"Discovery cycles" on page 285
A discovery cycle has occurred when the discovery data flow for a particular cycle has gone from start to finish. A full discovery might require more than one cycle.

Related tasks:

"Configuring EMS discoveries" on page 88
You can configure Network Manager to collect topology data from Element Management Systems (EMS) and integrate this data into the discovered topology.

Discovering device existence with collectors

During a collector discovery, the discovery of device existence takes place in several steps.

Figure 8 on page 296 shows how the initial existence of devices held on the collectors is discovered.

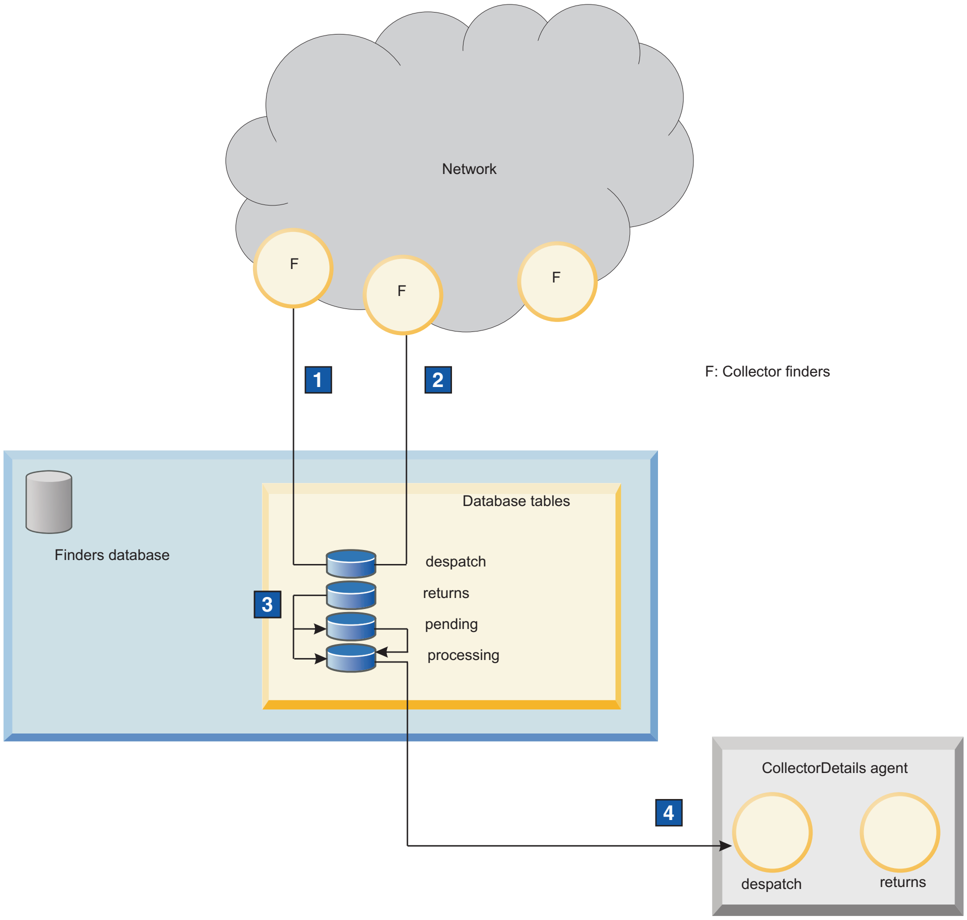

Figure 8. Collector discovery process flow: discovery of device existence

The following process flow describes Figure 8:

1 : The collector finders receive instructions from its configuration files and then proceeds to the network to look for collectors.

2 : The collector finders return the list of devices to the finders.returns table.

3 : Immediately after the device existence information is placed into the finders.returns table, the FnderRetProcessing stitcher moves the information to the finders.processing table, to denote that the network entity is being processed. If the discovery is in the blackout state, the information is placed into the finders.pending table.

4 : The FnderProcToDetailsDesp stitcher moves the information about device existence from the finders.processing table to the CollectorDetails.despatch table so that the CollectorDetails agent can process the information.

Discovering basic device information

During a collector discovery, the discovery of basic device information takes place in several steps.

The following figure shows how basic device details are discovered in a collector discovery.

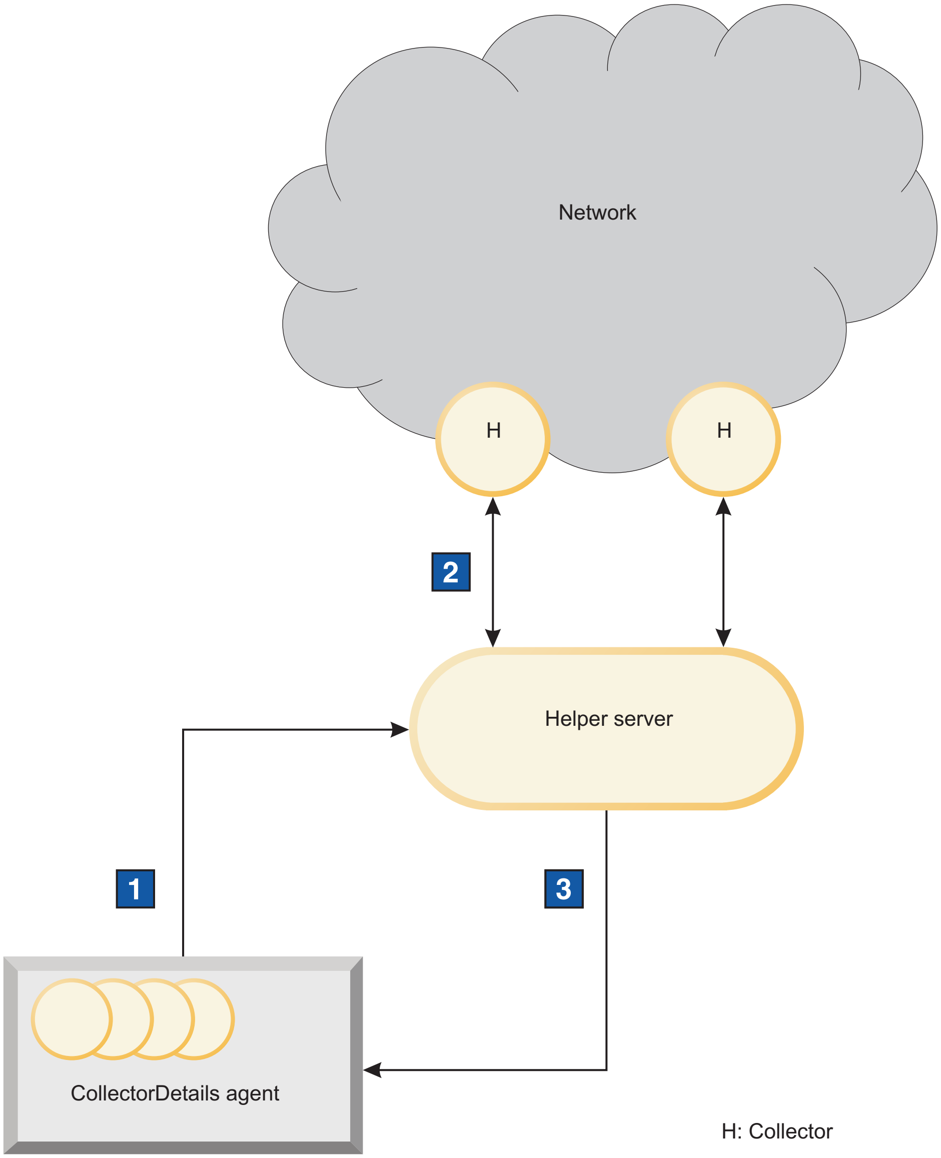

Figure 9. Collector discovery process flow: Discovery of basic device information

The following process flow describes Figure 9:

1 : All the agent despatch tables are active, so an insertion into the CollectorDetails.despatch table automatically triggers the CollectorDetails agent to discover basic device information from the collector.

2 : The CollectorDetails agent uses the Helper Server to interrogate the helper collector .

3 : The information retrieved from the network is returned to the CollectorDetails.returns table.

Discovering detailed device information

During a collector discovery, the discovery of detailed device information takes place in several steps.

The following figure shows how detailed device information is discovered in a collector discovery.

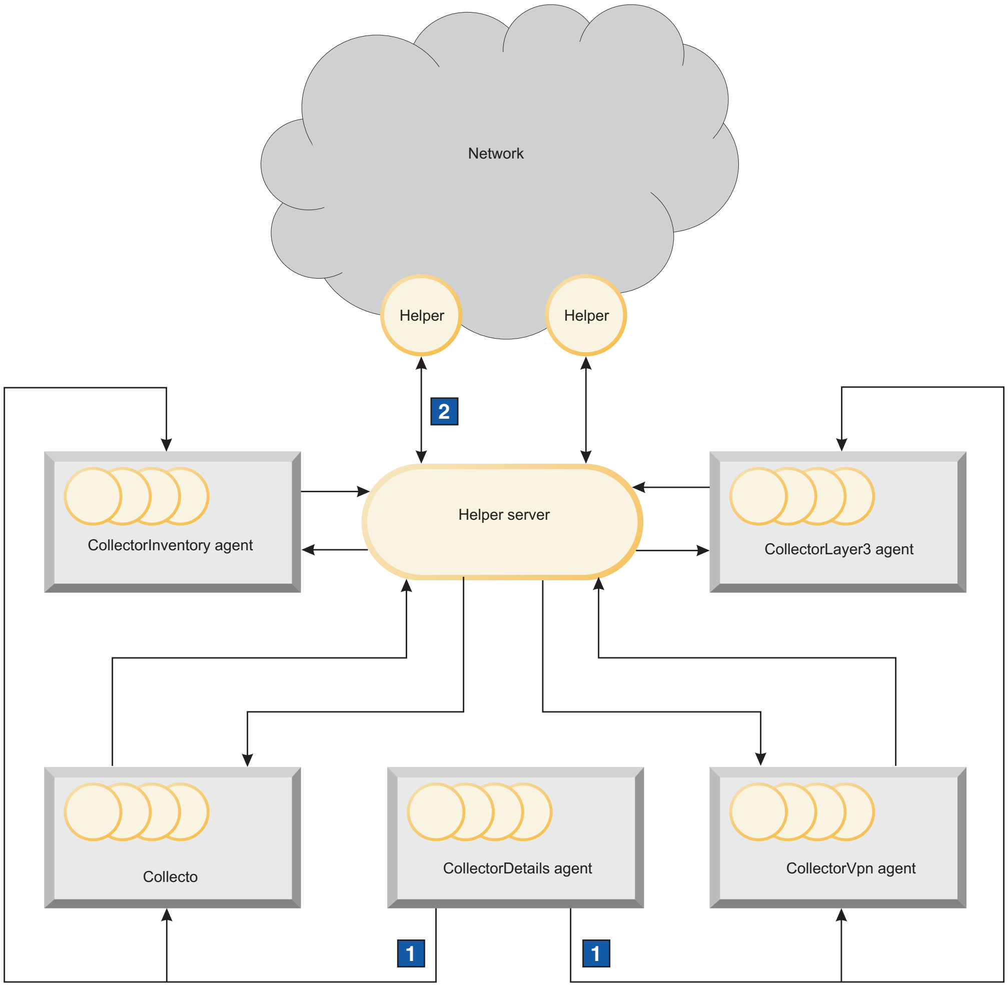

Figure 10. Collector discovery process flow: detailed device information

The following process flow describes Figure 10:

1 : The CollectorDetailsRetProcessing stitcher passes the information in the CollectorDetails.returns table to the despatch table of the following collector agents for processing: the CollectorInventory agent, the CollectorLayer2 agent, the CollectorLayer3 agent, and the CollectorVpn agent

2 : Inserting information into the despatch table of an agent triggers an attempt by that agent to discover information about that device. The collector agents interrogate the collectors to discover the following information about each device. The CollectorInventory agent discovers local interface, Entity MIB-style information, and IP addresses associated with the device. The CollectorLayer2 agent gathers information for each resolved layer 2 connection of each processed device. The CollectorLayer3 agent gathers information for each resolved layer 3 connection of each processed device. The CollectorVpn agent gathers VPN information for each processed device.

Rediscovery

When a discovery has completed, **ncp_disco** enters rediscovery mode, in which the discovery of new devices results in updates to the topology model.

Full and partial rediscovery

By modifying the stitchers, you can configure the way DISCO treats devices that are found in the rediscovery mode.

By default, when the system is in rediscovery mode and either a new device is found or an existing device changes, the device is rediscovered. The stitchers ensure that the device is rediscovered only once. The stitchers also check that the change has not caused the relationship of the device with its neighbors to change. If necessary, the neighbors of the device are rediscovered. If the number of devices that need to be rediscovered as a result of relationship changes exceeds a certain limit, the rediscovery process initiates a full rediscovery.

Related concepts:

"About types of discovery" on page 1
Different terms are used to describe network discovery, depending on what is being discovered and how the discovery has been configured. You can run discoveries, rediscoveries, full and partial discoveries, and you can set up automatic discovery.

Process flow of the FnderRetProcessing stitcher

To configure the way in which DISCO handles newly discovered devices, edit the FnderRetProcessing.stch stitcher. This stitcher processes the entries that are placed into the finders.returns table.

The default process flow of the FnderRetProcessing.stch stitcher is:

1. When an entry is placed in the finders.returns table, the stitcher checks whether the device is in the scope of the discovery. If the device is not in scope, it is ignored.
2. If the device is in scope and disco.status.m_DiscoveryMode=0, that is, DISCO is in discovery mode, the stitcher moves the device details to either the finders.pending table to be processed later (if the discovery is in the blackout state) or the finders.processing table to be processed now.
3. If the device is in scope and disco.status.m_DiscoveryMode=1, that is, DISCO is in rediscovery mode, the stitcher determines whether the device needs to be rediscovered. By default, the stitcher rediscovers:
   - Devices for which finders.returns.m_Creator='Rediscovery'. There is no Rediscovery finder, but this column is set to 'Rediscovery' by other stitchers, such as ProcRemoteConns.stch, to indicate that as a result of rediscovering other devices this device needs to be rediscovered.
   - Any newly found device that is in scope and has not already been discovered.
4. If necessary, you can alter the section of the FnderRetProcessing.stch stitcher that performs the above checks in order to configure when rediscovery of a device takes place, although this configuration adjustment must be undertaken by advanced users only.
5. If a device that has already been discovered is to be rediscovered, the stitcher refreshes the information held in the Helper Server that relates to that device.

6. For all devices to be rediscovered, the stitcher removes the old entries for that device from the finders.processing, Details.returns and Details.despatch tables, copying the old information to the rediscoveryStore.dataLibrary table for comparison purposes.
7. The stitcher then places the details of the device to be rediscovered into the finders.processing table and the FnderProcToDetailsDesp.stch stitcher moves the device details to the Details agent.

Processing information from discovery agents during rediscovery

After the entity that is being rediscovered has been processed by the Details agent, and the details are placed in the Details.returns table, the DetailsRetProcessing.stch stitcher compares the old data in the rediscoveryStore.dataLibrary table with the new data. By default, the rediscovery continues from this point.

If necessary, you can edit the DetailsRetProcessing.stch stitcher so that rediscovery continues only when certain conditions are in place. For example, rediscovery continues only when SNMP access is available.

The rediscovery data is processed by the AssocAddress agent and then by the appropriate agents (according to the configured discovery process flow) and sent to their returns tables.

A full discovery combines the information from the discovery agent returns tables to produce the topology. However, in a rediscovery, the information must be checked to determine whether the relationships between devices have changed as a result of the new information.

For example, if the device being rediscovered, device A, was connected to device B before the rediscovery, but is now connected to a third device, device C, then both B and C must also be rediscovered because their relationship has changed. The AgentRetProcessing.stch stitcher determines the relationships between devices and the comparison is done by ProcRemoteConns.stch. Switches and hubs need to be rediscovered differently to routers because the connectivity information they provide is indirect instead of direct. Any entity that also needs to be rediscovered as a consequence of rediscovery is inserted back into the finders.returns table with the parameter m_Creator='Rediscovery'.

Full rediscoveries

Comparing the current relationship between devices to their previous relationship, and rediscovering all the devices whose relationships have changed, can sometimes become circular. However, the discovery process includes a check to prevent this repetition.

If the ratio of entities that have been compared to the entities that need to be rediscovered exceeds the percentage specified in the disco.config.m_PendingPerCent column, then DISCO stops rediscovering individual devices and initiates a full network discovery.

Additionally, the fact that all rediscovered entities are recorded in the rediscoveryStore.rediscoveredEntities table means that a given entity can be rediscovered only once.

Rediscovery completion

When all the entities that need to be rediscovered have been processed, the topology layers are rebuilt by the FinalPhase.stch stitcher. This stitcher also clears the rediscoveryStore database so that it is ready for the next rediscovery.

It is important to note that DISCO might go through many discovery cycles during rediscovery before the topology is rebuilt. DISCO rebuilds the topology **only** when there are no entities needing to be rediscovered.

Option to rebuild topology layers

You can specify whether to rebuild the topology layers following a partial rediscovery. Using this option, you can increase the speed of partial rediscovery.

Suggested reasons to rebuild or not to rebuild the topology layers are:

- If you specify that the topology layers should *not* be rebuilt following partial rediscovery, the result is that new devices are added to the topology much faster than they would be if the topology layers are rebuilt; however, the resulting topology may not be complete. Connectivity associated with the newly discovered device is not fully reflected in the topology. Topology layers are fully rebuilt when a full rediscovery is run.
- If you specify that topology layers *should* be rebuilt following partial rediscovery, the result is an accurate topology showing all connectivity. However the process of adding new devices takes longer.

Use the m_RebuildLayers field in the disco.config table to specify whether or not to rebuild topology layers following partial rediscovery. You set this value as follows:

- If disco.config.m_RebuildLayers=0, then following partial rediscovery, topology layers stitchers are not run. The result is a much quicker partial discovery: however, connectivity associated with the newly discovered device is not fully reflected in the topology.
- If disco.config.m_RebuildLayers=1, then following partial rediscovery, topology layers stitchers are run. Partial rediscovery takes longer but results in a complete topology.

Appendix C. Discovery agents

Use this information to support the selection of discovery agents to run as part of your discovery.

The following topics provide information on the discovery agents available. There is also guidance on the agents to select, based on the characteristics of your network.

Agents

Discovery agents retrieve information about devices in the network. They also report on new devices by finding new connections when investigating device connectivity. Discovery agents are used for specialized tasks. For example, the ARP Cache discovery agent populates the Helper Server database with IP address-to-MAC address mappings.

In addition to the main discovery agents, which can be enabled or disabled according to your discovery requirements, there are two agents that must always be run: the Details agent and the Associated Address agent.

Each discovery agent has its own database resident within DISCO. These databases are generically structured and based on a template called the agentTemplate database.

Each discovery agent database contains the following tables:

- *agentName*.despatch
- *agentName*.returns

Note: The default configuration sets the majority of agents to run. This is because the more agents that are run, the wider the range of networks that can be discovered. Furthermore, agents are designed to quickly stop analyzing devices that do not provide the data they require. This means that running a large number of agents increases network traffic by a very small amount only.

Note: Network Manager kills all discovery agents at the end of data collection stage 3. This ensures that the next discovery restarts the agents and forces the agents to reread their configuration files at the beginning of a discovery, thereby detecting any changes to the configuration files.

Related reference:

"Subprocess databases" on page 220
The finders, Details, and agent databases are used during the discovery by the discovery engine subprocesses to store information retrieved from the network. The databases are defined within the configuration file, DiscoSchema.cfg.

"DiscoAgentReturns.filter configuration file" on page 55
The DiscoAgentReturns.filter configuration file allows you to apply a topology data filter to data returned by all discovery agents.

Details agent

This agent is triggered by the entries in the finders.processing table. At least one finder is needed to activate this agent. The SNMP helper configuration for associated devices is also a prerequisite for running this agent.

The Details agent retrieves basic information about devices discovered by the finders, and determines whether SNMP access to the device is available. This mandatory agent is triggered by the entries in the finders.processing table, so at least one finder is needed to activate this agent.

The Details agent is triggered when device information (usually transferred from the finders by a stitcher) is placed in the Details.despatch database table.

The Details agent retrieves basic information from the network and deposits this information in the Details.returns table. The basic information retrieved comprises the DNS name of the device obtained by the configured DNS helper, and the system object ID obtained by the SNMP helper. IpForwarding data is downloaded and inserted into the ExtraInfo field which is used to identify routing devices. SysName information is also downloaded for use if this optional naming scheme is required. The insertion of data into the returns table triggers a stitcher that sends this information to the Associated Address agent.

Related concepts:

"Discovering device details (standard)" on page 287
The standard discovery of device details is carried out in several steps.

Associated Address (AssocAddress) agent

This mandatory agent is triggered by the output of the Details agent. The SNMP helper configuration for associated devices is a prerequisite for running this agent.

When an interface on a device has been discovered, and basic device information has been retrieved by the Details agent, a stitcher passes the discovered device information to the Associated Address agent. This agent downloads all the other IP addresses associated with the device and adds them to a central registry, held in the translations.ipToBaseName table, provided the device details are not already in the registry. Downloading all the associated IP addresses ensures that any given device is only interrogated once by the main discovery agents, thus reducing the load on the agents. Any attempt to discover a device more than once (using multiple interfaces) is blocked by the Associated Address agent as the device details are already in the translations database.

Provided the device being checked has not already been discovered, a stitcher sends the device details to the appropriate discovery agent for the retrieval of device connectivity and protocol-specific information.

Related concepts:

"Discovering associated device addresses" on page 289
There are several steps in the process flow during the discovery of associated device addresses.

Interface data retrieved by agents

The Interfaces agent downloads interface information from the interfaces table of RFC1213.mib. For each device discovered, the interface information is written to the m_LocalNbr field within each record in the relevant agent.returns table.

The Interfaces agent downloads interface information primarily from the interfaces table of RFC1213.mib. For each device discovered, the interface information is written to the m_LocalNbr field within each record in the relevant agent.returns table. The interface information can hold a number of subfields, including an index number that identifies the interface together with the properties of that interface and values for each property. For example, the m_LocalNbr field may include the following subfields:

- m_LocalNbr->m_IfIndex: the index associated with this interface
- m_LocalNbr->m_IfType: the type of interface
- m_LocalNbr->m_SubnetMask: the subnet mask of the interface

Related reference:

"Connectivity at the layer 3 network layer" on page 316
There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

Discovery agent definition file keywords

Discovery agent definition file keywords are used to define the operation of discovery agents.

DiscoAgentClass

The `DiscoAgentClass` keyword specifies the basic type of agent. The following table identifies the most commonly used values:

Value	Description
0	Specifies an IP type agent.
1	Specifies a switch type agent.
2	Specifies a hub type agent.
3	Specifies an ATM device type agent.
4	Specifies an FDDI type agent.
5	Specifies a PVC type agent.
6	Specifies a frame relay type agent.
8	Specifies a NAT gateway agent.

The following example shows a `DiscoAgentClass` keyword set to a frame relay type agent. Frame relay type agents typically discover Frame Relay interfaces and connections between two points on Frame Relay networks that incorporate specific network devices, for example, CISCO devices.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentClass( 6 );
.
.
.
}
```

DiscoAgentClassEnabledByDefault

The `DiscoAgentClassEnabledByDefault` keyword specifies whether the agent is enabled by default for full discoveries. Specify one of the following values:

Value	Description
0	Specifies that the agent is not enabled by default for full discoveries.
1	Specifies that the agent is enabled by default for full discoveries.

The following example shows a `DiscoAgentClassEnabledByDefault` keyword set to enable a frame relay type agent by default for full discoveries.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentClass( 6 );
.
.
.
DiscoAgentEnabledByDefault( 1 );
}
```

DiscoAgentClassEnabledByDefaultOnPartial

The `DiscoAgentClassEnabledByDefaultOnPartial` keyword specifies whether the agent is enabled by default for partial discoveries. Specify one of the following values:

Value	Description
0	Specifies that the agent is not enabled by default for partial discoveries.
1	Specifies that the agent is enabled by default for partial discoveries.

The following example shows a `DiscoAgentClassEnabledByDefaultOnPartial` keyword set to enable a frame relay type agent by default for partial discoveries.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentClass( 6 );
.
.
.
DiscoAgentEnabledByDefaultOnPartial( 1 );
DiscoAgentEnabledByDefault( 1 );
}
```

DiscoAgentIsIndirect

A direct agent returns relationship data about objects that it is directly connected to at the layer it deals with. An indirect agent returns relationship data about objects it is indirectly connected to. The most common indirect agents are switch agents. The remote neighbor records for indirect agents relate to devices that can be reached from a specific port, not from devices to which they are directly connected. The relationship data from indirect agents is required to determine which remote neighbor records of a device need to be rediscovered when the device changes.

The DiscoAgentIsIndirect keyword specifies whether the agent is an indirect agent that returns relationship data about objects it is indirectly connected to. Specify one of the following values:

Value	Description
0	Specifies that the agent is a direct agent.
1	Specifies that the agent is an indirect agent.

The following example shows a DiscoAgentIsIndirect keyword set to specify that a frame relay type agent is a direct agent.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentGUILocked( 0 );
DiscoAgentClass( 6 );
DiscoAgentIsIndirect( 0 );
.
.
.
DiscoAgentEnabledByDefaultOnPartial( 1 );
DiscoAgentEnabledByDefault( 1 );
}
```

DiscoAgentCompanionAgents

The DiscoAgentCompanionAgents keyword is used to display in the GUI the agent or agents that this agent should execute with.

The following example shows a DiscoAgentCompanionAgents keyword that displays in the GUI the agent (ArpCache.agnt) that should execute with the Centillion Networks agent.

```
DiscoCompiledAgent
{
.
.
.
-- This agent examines all devices originally made by Centillion
-- Networks (enterprise OID 1.3.6.1.4.1.930), to see if it can
-- discover them.
.
.
.

DiscoAgentCompanionAgents( "ArpCache" );
.
.
.
}
```

DiscoAgentCompletionPhase

The DiscoAgentCompletionPhase keyword specifies during which of the discovery phases the specified agent should complete executing. Specify one of the following values:

Value	Description
1	Specifies that the agent should complete execution during discovery phase 1.
2	Specifies that the agent should complete execution during discovery phase 2.
3	Specifies that the agent should complete execution during discovery phase 3.

The following example shows a DiscoAgentCompletionPhase keyword set to enable a frame relay type agent to complete execution during discovery phase 1.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentCompletionPhase( 1 );
.
.
.
DiscoAgentEnabledByDefaultOnPartial( 1 );
DiscoAgentEnabledByDefault( 1 );
}
```

DiscoAgentConflictingAgents

The DiscoAgentConflictingAgents keyword is used to display in the GUI the agent or agents that this agent should not execute with.

The following example shows a DiscoAgentConflictingAgents keyword that displays in the GUI the agents (IpRoutingTable.agnt and IpForwardingTable.agnt) that should not execute with the IP backup routes agent.

```
DiscoCompiledAgent
{
.
.
.
-- This agent examines every device with SNMP access to see if it
-- can discover it.
.
.

DiscoAgentConflictingAgents( "IpRoutingTable","IpForwardingTable" );
.
.
.
}
```

DiscoAgentDescription

The DiscoAgentDescription keyword specifies a description of the agent that is displayed in the GUI.

The following example shows a DiscoAgentDescription keyword that specifies a description to display in the GUI for a frame relay type agent. The description makes use of HTML coding.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentDescription("
<b>Agent Name :</b>  CiscoFrameRelay<br>
<br>
<b>Agent Type :</b> Layer 3<br>
<br>
<b>Agent Prerequisites :</b>  SNMP helper configuration for associated devices.<br>
<br>
<b>Operation :</b><br>
Discovers Frame Relay interfaces and connections between two points on Frame Relay
networks that incorporate Cisco devices. If you need to add DLCI information to the
interfaces of Frame Relay devices, then run Frame Relay agents in conjunction with
the IP layer agents.<br>
<br>
 ");
.
.
.
}
```

DiscoAgentMinCertifiedDeviceOS

The DiscoAgentMinCertifiedDeviceOS keyword specifies a device operating system specific filter. This filter can be configured to run the specified agent against specific releases of a device operating system.

The following example shows a DiscoAgentMinCertifiedDeviceOS keyword that specifies a device operating system specific filter for an agent that discovers MPLS VRFs, VPNs, and label switching information from CISCO routers. This device operating specific filter configures the agent to run against the following CISCO devices and associated operating system releases:

- m_ObjectId — Specifies the CISCO devices (OID 1.3.6.1.4.1.9) that the agent attempts to discover.
- m_OSVersion — Specifies the CISCO device operating system filter that configures the agent to run against the following device operating system versions:
  - 12.0 releases of 12.0(27) or later which are not experimental
  - 12.2 releases of 12.2(19) or later which are not experimental
  - 12.3 releases of 12.3(18) or later which are not experimental
  - 12.4 releases

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentMinCertifiedDeviceOS
    (
      "(
          m_ObjectId LIKE '1\.3\.6\.1\.4\.1\.9\.',
          m_OSVersion >= '12.0(27)' AND m_OSVersion < '12.1' AND m_OSVersion
                         NOT LIKE '.*Experimental.*',
          m_MibVar = 'sysDescr.0'
        ),
        (
          m_ObjectId LIKE '1\.3\.6\.1\.4\.1\.9\.',
          m_OSVersion >= '12.2(19)' AND m_OSVersion < '12.3' AND m_OSVersion
                         NOT LIKE '.*Experimental.*',
          m_MibVar = 'sysDescr.0'
        ),
      (
          m_ObjectId LIKE '1\.3\.6\.1\.4\.1\.9\.',
          m_OSVersion >= '12.3(18)' AND m_OSVersion < '12.4' AND m_OSVersion
                         NOT LIKE '.*Experimental.*',
          m_MibVar = 'sysDescr.0'
       ),
      (
          m_ObjectId LIKE '1\.3\.6\.1\.4\.1\.9\.',
          m_OSVersion >= '12.4',
          m_MibVar = 'sysDescr.0'
        )"
    );
.
.
.
}
```

DiscoAgentPrecedence

The DiscoAgentPrecedence keyword specifies which agent gets precedence when there is conflicting data from two agents. Specify a value that is greater than or equal to 0 (zero). The recommended range of values is from 1 to 100, where the higher the number the higher the precedence. The higher the precedence the more that agent data is correct. For example, if given conflicting data from a precedence 2 agent and a precedence 3 agent then the precedence 3 agent data is used.

The following example shows a DiscoAgentPrecedence keyword for a frame relay type agent set to a precedence of 2.

```
DiscoCompiledAgent
{
.
.
.
DiscoAgentGUILocked( 0 );
DiscoAgentClass( 6 );
DiscoAgentIsIndirect( 0 );
DiscoAgentPrecedence( 2 );
.
.
.
DiscoAgentEnabledByDefaultOnPartial( 1 );
DiscoAgentEnabledByDefault( 1 );
}
```

DiscoPerlAgent

The DiscoPerlAgent keyword specifies whether this .agnt file refers to a Perl agent.

The following example shows a DiscoPerlAgent keyword specified for a Perl based agent that extracts information about the operating system running on the device.

```
DiscoPerlAgent
{
.
.
.
DiscoAgentGUILocked( 0 );
 DiscoAgentClass( 0 );
 DiscoAgentIsIndirect( 0 );
 DiscoAgentPrecedence( 2 );
 DiscoAgentEnabledByDefaultOnPartial( 0 );
 DiscoAgentEnabledByDefault( 0 );
}
```

Types of agents

The agents supplied with Network Manager can be divided into categories according to the type of data they retrieve or the technology they discover.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

Discovering connectivity among Ethernet switches

Discovery agents that discover connectivity information between Ethernet switches have three main operational stages: gain access to the switch and download all the switch interfaces; discover VLAN information for the switch; download the forward database table for the switch.

A list of the discovery agents that handle Ethernet switches is shown in Table 136.

Note: Before enabling these layer 2 agents, it is necessary to configure SNMP access. Some agents also require Telnet access and Telnet Helper configuration. This is specified where required.

Table 136. Ethernet switch discovery agents

Agent name	Function
AccelarSwitch	The AccelarSwitch agent contains the specialized methods for retrieving connectivity information from Accelar routing switches. These devices are now branded as the Nortel Passport 86xx series. This agent also discovers BayStack 450 and BayStack 470 devices. This agent downloads the switch forwarding database (FDB) table and the VLAN information for the device. The switch stitcher uses this information to resolve layer 2 Ethernet connectivity.

Table 136. Ethernet switch discovery agents (continued)

Agent name	Function
BayEthernetHub	The BayEthernetHub agent discovers hub cards manufactured by Bay. Connectivity information is downloaded from the hub and the connectivity is resolved by the HubFdbToConnections stitcher. Before enabling this agent, it is also necessary to configure the SNMP Helper.
CentillionSwitch	The CentillionSwitch agent contains the methods needed to retrieve and resolve information from the Centillion switching devices, in particular the enterprise-specific VLAN information.
ChipcomDistributedMM	The ChipcomDistributedMM agent discovers the Ethernet switch connectivity for 3Com CoreBuilder 5000 devices containing distributed management modules.
ChipcomEthernetMM	The ChipcomEthernetMM agent is appropriate for Chipcom online concentrators containing Ethernet Management Modules (EMMs), and discovers the Ethernet connectivity of Chipcom EMMs.
CiscoSRP	The CiscoSRP agent discovers the connectivity of networks using the Spatial Reuse Protocol (SRP), that is, DPT Ring topologies. SRP is a layer 2 protocol developed by Cisco that uses 'side' information to identify adjacent neighbours in its ring topology.The CiscoSRP agent discovers connectivity of any device that supports the CISCO-SRP-MIB. The agent definition file is configured by default to accept only Cisco devices with any IOS version, except those supported by the CiscoSRPTelnet agent. The agent only accepts devices that support the srpMacAddress MIB variable. IOS version 12.2(14)S7 and 12.2(18)S, used with NPE-G1 cards, are known to corrupt SNMP data.IOS version 12.2(15)BC1 is known to lack CISCO-SRP_MIB support.
CiscoSRPTelnet	The CiscoSRPTelnet agent discovers the connectivity of networks using the Spatial Reuse Protocol (SRP), that is, DPT Ring topologies. SRP is a layer 2 protocol developed by Cisco that uses 'side' information to identify adjacent neighbours in its ring topology. The CiscoSRPTelnet agent discovers the connectivity of any device that supports the **show controllers srp** command. The agent definition file is configured to only accept Cisco devices that have an IOS known not to support the CISCO-SRP-MIB and those IOS versions that have known issues with SNMP discovery.IOS version 12.2(14)S7 and 12.2(18)S, used with NPE-G1 cards, are known to corrupt SNMP data.IOS version 12.2(15)BC1 is known to lack CISCO-SRP_MIB support. **Note:** Before enabling this agent, it is necessary to configure Telnet access and the Telnet Helper.

Table 136. Ethernet switch discovery agents (continued)

Agent name	Function
CiscoSwitchSnmp	The CiscoSwitchSnmp agent contains the specialized methods for retrieving information from Cisco switches using SNMP. This agent uses a variety of methods for finding VLANs and card or port to ifIndex mappings because different Cisco switches store this information in different MIB variables. When discovering devices using SNMPv3, the Cisco switches must have the VLAN context added to the view group for each VLAN.
CiscoSwitchTelnet	The CiscoSwitchTelnet agent contains specialized methods for retrieving connectivity information from Cisco switches using Telnet. This agent uses a variety of methods to find VLANs and card/port to ifIndex mappings because different Cisco switches store this information in different MIB variables. Only FDB tables are downloaded using Telnet. All other information is downloaded using SNMP. The Telnet commands used to obtain the FDB table are **show cam dynamic** and **show mac-address table**. Some devices might require enable mode in order to run the **show mac-address table** command. **Note:** Before enabling this agent, it is necessary to configure SNMP and Telnet access and their respective Helpers.
CiscoVSS	The Cisco VSS agent discovers Virtual Switching System information from Cisco switches.
Corebuilder3ComSwitch	The Corebuilder3ComSwitch agent discovers links for the CoreBuilder 9000 layer 3 switches manufactured by 3Com.
DasanSwitchTelnet	The DasanSwitchTelnet agent is responsible for the discovery of the layer 2 connectivity held in the FDB/MAC table of Dasan switches. The agent was developed against the following devices:V5208 (OS 9.07)V5224 (OS 9.10)The agent is able to discover layer 2 connectivity, VLANs and trunk ports. It is configured to only run against devices with a sysObjectID of 1.3.6.1.4.1.6296.* and that support the command **show vlan**. **Note:** Before enabling this agent, it is necessary to configure Telnet access and the Telnet Helper.
DefaultEthernetHub	This agent has a specialized class for dealing with semi-intelligent hubs.
EnterasysSwitch	The EnterasysSwitch agent provides layer 2 connectivity discovery by retrieving the FDB table and VLAN information from the device. The agent discovers layer 2 connectivity for devices that support the IEEE 802.1q or IEEE 802.1d standards, as modelled in the Q-BRIDGE-MIB and BRIDGE-MIB SNMP MIBs respectively. **Note:** This agent is used for Enterasys devices that do not have SecureFast turned on.

Table 136. Ethernet switch discovery agents (continued)

Agent name	Function
ExtremeSwitch	The ExtremeSwitch agent obtains layer 2 connectivity information, EDP neighbours, and VLAN details from Extreme switches. The Extreme devices must be configured to enable SNMP access and dot1dFdbTable population to achieve a detailed layer 2 discovery. Send the following commands to each Extreme device: • **enable snmp access** • **enable dot1dFdbTable** This configuration change is only required on switches running a version of ExtremeWare® prior to 6.1.8.
FoundrySwitch	The FoundrySwitch agent discovers switch connectivity of any Foundry device that supports the IEEE 802.1q or IEEE 802.1d standards, as modelled in the Q-BRIDGE-MIB and BRIDGE-MIB SNMP MIBs respectively. The agent definition file is configured to accept all SNMP-enabled Foundry devices by default. The agent will only discover devices that support the Q-BRIDGE-MIB dot1qVlanVersionNumber MIB variable or the BRIDGE-MIB.The FoundrySwitch agent also obtains multislot port trunking information, but does not discover single-slot port trunking.Some Foundry devices only support IEEE 802.1d and, as a consequence, no VLAN information is discovered for these devices.
HuaweiSwitchTelnet	The HuaweiSwitchTelnet agent discovers the Ethernet switch connectivity for Huawei Quidway switches. This agent is Telnet-based, but also requires SNMP access to discover certain information. It requires completion of the Privileged mode (Super 3 mode) sections of the TelnetStackPasswords.cfg configuration file. Failure to complete these sections will result in the agent failing. Certain Telnet commands have the side-effect of changing the command prompt of a Huawei device. For example, the command prompt: `<device_name>` becomes `[device_name]` or `[device_name-diag]` when certain commands are issued. It is essential that the parameters m_ConPrompt and m_PrivConPrompt in TelnetStackPasswords.cfg are configured to cope with these variations. **Note:** Before enabling this agent, it is necessary to configure Telnet access and the Telnet Helper.
HPSwitch	The HPSwitch agent contains the specialized methods for retrieving connectivity information from HP ProCurve switches, including the download of enterprise-specific VLAN information.

Table 136. Ethernet switch discovery agents (continued)

Agent name	Function
Marconi3810	The Marconi3810 specialised agent discovers the Ethernet connectivity of Marconi ES-3810 switches running operating system version 4.x.x and 5.x.x. This agent also removes connectivity from LANE interfaces by default - this can be configured using the GetElanData flag in the .agnt file.
NortelSwitch	The NortelSwitch agent retrieves Layer 2 connectivity information, including Split Level Multi-Trunking (SMLT) information, from Nortel switches.
SecureFast	The SecureFast agent contains the specialized methods for retrieving connectivity information from Enterasys/Cabletron SecureFast VLAN switches. These devices use the Cabletron Discovery Protocol to discover their neighbours and have SecureFast operating mode turned on. This agent is sent to all Cabletron and Enterasys devices, specified by 1.3.6.1.4.1.52.* and 1.3.6.1.4.1.5624.* in the .agnt file, and determines whether a device is SecureFast enabled by downloading the sfpsCommonNeighborSwitchMAC MIB variable. Devices in SecureFast mode do not support the dot1dBridge MIBs.
StandardSwitch	The StandardSwitch generic agent provides layer 2 connectivity discovery of all switches for which a specialized agent does not exist. The agent discovers layer 2 connectivity for devices that support the IEEE 802.1q or IEEE 802.1d standards, as modelled in the Q-BRIDGE-MIB and BRIDGE-MIB SNMP MIBs respectively. Devices in SecureFast mode do not support the dot1dBridge MIBs.
SuperStack3ComSwitch	The SuperStack3ComSwitch agent finds the connections in stacked switches manufactured by 3Com.
XyplexEthernetHub	The XyplexEthernetHub agent discovers the layer 2 connectivity of intelligent hubs manufactured by Xyplex.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

"DiscoTelnetHelperSchema.cfg configuration file" on page 71
The DiscoTelnetHelperSchema.cfg configuration file defines the operation of the Telnet helper, which returns the results of a Telnet operation into a specified device.

Connectivity at the layer 3 network layer

There are a number of discovery agents that retrieve connectivity information from OSI model layer 3 (the *Network Layer*). Layer 3 is responsible for routing, congestion control, and sending messages between networks.

Table 137. Layer 3 network layer agents

Agent name	**Function**
AlteonVRRP	VRRP is not modelled for RCA. The AlteonVRRP agent only sets tags on VRRP interfaces that show the state of Alteon routers at the time of the discovery. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
CiscoBGPTelnet	The CiscoBGPTelnet agent downloads the following BGP data from Cisco routers: • Peer data: the agent retrieves iBGP and eBGP data from peer routers. • Route data: the agent retrieves routing information from BGP routing tables of peer routers. This option is off by default as it will retrieve huge amounts of data from a typical service provider network. This agent also provides the option to configure a filter to specify the route data that you would like to retrieve. **Note:** Before enabling this agent, configure Telnet access and the Telnet helper.
CiscoFrameRelay	The CiscoFrameRelay agent discovers Frame Relay interfaces and connections between two points on Frame Relay networks that incorporate Cisco devices. Frame Relay agents should be run in conjunction with the IP layer agents if you want to add DLCI information to the interfaces of Frame Relay devices. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
CiscoOSPFTelnet	The CiscoOSPFTelnet agent is responsible for discovery of Cisco devices running the Open Shortest Path First (OSPF) protocol. This agent provides complementary information to that of the StandardOSPF agent, such as what OSFP processes are running and virtual-link information. **Note:** Before enabling this agent, configure Telnet access and the Telnet helper.
ExtremeESRP	The ExtremeESRP agent discovers Extreme Standby Routing Protocol (ESRP) information from Extreme routing switches. ESRP is a feature of ExtremeWare that allows multiple switches to provide redundant routing services to users. The agent relies on the `extremeEsrpTable` and `extremeEsrpNeighborTable` of the EXTREME-ESRP-MIB being correctly populated. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.

Table 137. Layer 3 network layer agents (continued)

Agent name	Function
FoundryVRRP	VRRP is not modelled for RCA. The FoundryVRRP agent only sets tags on VRRP interfaces that show the state of Foundry routers at the time of the discovery. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
HSRPSnmp	The HSRPSnmp agent uses SNMP to retrieve information from routing devices that use the HSRP (Hot Stand-by Routing Protocol) Virtual IP protocol. The HSRPSnmp agent retrieves data on primary and secondary HSRP IP addresses, which is used for interface discovery and visualization. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
InetRouting	The InetRouting agent discovers connectivity.
Interfaces	This agent is triggered by the AssocAddress agent returns. The Interfaces agent downloads interface information primarily from the interfaces table of RFC1213.mib. The information will then be written to the m_LocalNbr field of the returned entities. You can increase or decrease the number of returned variables by modifying the Interfaces.agnt. Any basic MIB variable (sysDescr, sysName, and so on) or MIB variable that is indexed by the ifIndex can be added to the OIDs to download in the .agnt file. The Interfaces agent also retrieves IPv6 interface information. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
IpBackupRoutes	The IpBackupRoutes agent finds links by looking through the IpNetToMedia MIB table, which gives the physical and IP address of devices connected to the router. This agent is not enabled by default because it retrieves a large amount of information that is not essential in order to determine layer 3 connections. Furthermore, this information may be obsolete because it is downloaded from a table that is not dynamic and requires manual refresh. If you are performing a layer 2 discovery, then the server connectivity that this agent discovers is often obsolete, as it may have been superseded by switch connectivity information. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
IpForwardingTable	The IpForwardingTable agent finds links in the more recent version of the routing tables, that is, the IP Forwarding table as specified in RFC 2096. It also exploits Open Shortest Path First (OSPF) information to enhance the discovery of Juniper devices. This agent downloads elements from the routing table based on discovery scoping. The default setting assumes that the SNMP agent for a particular device supports partial matching. If the device cannot partial match, this should be specified in the DiscoRouterPartialMatchRestrictions section of the .agnt file. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
IpRoutingTable	Retrieves generic connectivity information by looking through the router routing table, as specified in RFC1213. The agent downloads elements from the routing table based on discovery scoping. The default agent setting assumes that the SNMP agents for particular devices support partial matching. If a device cannot partial match, this should be specified in the DiscoRouterPartialMatchRestrictions section of the .agnt file. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
ISISExperimental	Discovers connectivity between routers that support the experimental ISIS MIBs. This agent should be used when some of your routers are configured with netmasks of 255.255.255.255, making them unsuitable for standard discovery. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
LinkStateAdvOSPF	Retrieves link state advertisements (LSAs) from OSPF routers. These LSAs are used by the CreateOSPFNetworkLSAPseudoNodes stitcher to create OSPF pseudonodes. Pseudonodes overcome the problem of full meshing when representing OSPF area in Topoviz Network Views and enables connections within OSPF areas to be visualized in a clear, uncluttered manner.

Table 137. Layer 3 network layer agents (continued)

Agent name	Function
JuniperBGPTelnet	Downloads BGP information from Juniper routers. It is not enabled by default because it gathers a very specific piece of information only, that is, whether devices are route reflectors. **Note:** Before enabling this agent, configure Telnet access and the Telnet helper.
JuniperMXGroupTelnet	The JuniperMXGroupTelnet agent uses Telnet to discover logical collection information for Routing Engine Groups on Juniper MX devices.
NetScreenInterface	The NetScreenInterface agent retrieves information about all configured interfaces in Juniper Netscreen devices. The agent retrieves information about logical interfaces and other interfaces, which is not available from the standard IF-MIB, and requires both the NETSCREEN-INTERFACE-MIB.mib and NS-VPN-MON.mib files. The agent also retrieves VPN and tunnel connectivity information that is configured in Juniper NetScreen devices. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
NetScreenIpRoutingTable	The NetScreenIpRoutingTable agent retrieves information on IP routing tables configured on Netscreen devices. The agent determines the interfaces and sub-interfaces from the interface index of the Netscreen device. This agent performs the same function as the IpRoutingTable agent, but for Netscreen devices only, in order to take account of sub-interfaces which would not be discovered correctly by the IpRoutingTable agent. The NetScreenIpRoutingTable agent uses the IP-FORWARD-MIB Standard MIB and the NETSCREEN-INTERFACE-MIB. **Note:** The IpRoutingTable agent does not process the Netscreen devices processed by the NetScreenIpRoutingTable agent.
NokiaVRRP	Downloads VRRP information from routers that support the Nokia interpretation of the VRRP MIB. The information retrieved includes the VRRP state, ID, primary IP and associated addresses. This information is retrieved from the following MIB variables: • vrrpOperState • vrrpOperMasterIpAddr • vrrpAssoIpAddrRowStatus **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
NortelPassport	The NortelPassport agent retrieves Layer 3 connectivity and containment information from Nortel Passport switches.
RFC2787VRRP	The RFC2787VRRP agent downloads Virtual Router Redundancy Protocol (VRRP) information from routers that run RFC2787-compliant VRRP and support the RFC2787 VRRP MIB. Some Nokia firewalls support this MIB. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper. VRRP is not modelled for RCA. This agent sets tags on VRRP interfaces that show the state of the interfaces at the time of the discovery. The agent also downloads associated IP addresses, which are used to build VRRP collections. **Tip:** There are two subtly different versions of the VRRP MIB. They contain the same names but with different OIDs. If this agent does not work, use the other version of the VRRP MIB.

Table 137. Layer 3 network layer agents (continued)

Agent name	Function
StandardBgp	The StandardBgp agent is responsible for discovery of networks running the Border Gateway Protocol. It supports any device that complies with the standard RFC1657 (BGP4-MIB) MIB and discovers the following information: • Autonomous System IDs • BGP Peer connections to external peers (EBGP) • BGP Peer connections to internal peers (IBGP) • BGP acquired route data (not recommended) The agent definition file is configured to accept all SNMP enabled devices by default, but the agent will only accept devices that support the BGP44-MIB, bgpIdentifier MIB variable. The agent has the following additional configuration parameters in the DiscoAgentDiscoveryScoping section of its .agnt file: • GetPeerData – determines whether the agent should acquire BGP Peer data (activated by default). • GetRouteData – determines whether the agent should acquire BGP routes (deactivated by default). This may result in a large amount of data being discovered. The StandardBgp agent does not currently support peer groups, confederations, per VRF BGP processes, or route reflection. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper. It is also necessary to configure the Ping helper.
StandardOSPF	The StandardOSPF agent is responsible for the discovery of networks running the Open Shortest Path First (OSPF) protocol. It will support any device that complies with the standard RFC1850. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.
TraceRoute	The TraceRoute agent finds links by tracing the route taken by an ICMP ping packet with a predetermined life span. If you are using this agent, you should increase the value of m_Timeout in the DiscoPingHelperSchema.cfg configuration file, as traceroute functionality takes longer than standard ICMP. This agent is not enabled by default as it does not only operate on SNMP-enabled devices. Therefore, if this agent were switched on by default, it would trace the route to every device on the network. The result could be incomplete connectivity in a meshed environment or inaccurate connectivity in a load-balanced environment. **Note:** Before enabling this agent, configure SNMP access and the SNMP helper.

Related tasks:

"Configuring device access" on page 23
Specify SNMP community strings and Telnet access information to enable helpers and Network Manager polling to access devices on your network.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

"DiscoPingHelperSchema.cfg configuration file" on page 62
The DiscoPingHelperSchema.cfg configuration file defines how devices are to be pinged.

Topology data stored in an EMS

There are several discovery agents that retrieve information about devices managed by an EMS.

The routing protocol discovery agents query an EMS collector for basic and detailed information about devices managed by EMS. These agents are shown in Table 138.

Table 138. Routing protocol discovery agents

Agent name	Function
CollectorDetails	Retrieves basic information about the devices on the collector, including sysObjectId, sysDescr, and naming data.
CollectorInventory	Retrieves local neighbor, entity and associated address data for each of the devices on the collector
CollectorLayer2	Retrieves layer 2 connectivity information for the devices on the collector.
CollectorLayer3	Retrieves layer 3 connectivity information for the devices on the collector.
CollectorVpn	Retrieves layer 2 and layer 3 VPN data for the devices on the collector.

Related concepts:

"Components of the EMS integration" on page 90
EMS integration is composed of several components that assist in the collection of topology data.

Discovering connectivity among ATM devices

Asynchronous Transfer Mode (ATM) is an alternative switching protocol for mixed format data (such as pure data, voice, and video). Several types of discovery agents can be used to discover ATM devices on a network.

Note: Before enabling these agents, it is necessary to configure SNMP access and the SNMP Helper.

Table 139. ATM discovery agents

Agent name	Function
AtmForumPnni	The AtmForumPnni agent retrieves connectivity information from ATM devices that use the Private Network-to-Network Interface (PNNI) dynamic routing protocol and the ATM Forum's PNNI MIB. The PNNI protocol is commonly used on large networks, as it provides ATM switches with a detailed map of the network topology so that the ATM devices can make optimal routing decisions.
CellPath90	The CellPath90 agent enables discovery of the ATM connection of Marconi CellPath 90 WAN (Wide Area Network) multiplexers. The CellPath 90 WAN multiplexer does not know the ATM addresses of its neighbours, so it can only be discovered when it is connected to another, more intelligent, certified ATM device. The CellPath90 discovery agent is used in the calculation of network topology. It places information about the CellPath 90 into the correct layers within the discovery database.
CiscoPVC	The CiscoPVC agent retrieves PVC data from Cisco devices.
CiscoSerialInterface Telnet	The CiscoSerialInterfaceTelnet agent uses Telnet to retrieve Asynchronous Transport Mechanism (ATM) connectivity information from Cisco devices. Use this agent if you have Cisco routers that are connected by serial interfaces configured as ATM Private Virtual Circuits (PVCs). You must run the Interface agent with the CiscoSerialInterfaceTelnet agent.
ILMI	The ILMI agent retrieves connectivity information from devices using the Interim Local Management Interface (ILMI), an RFC standard for managing ATM and IP networks. It investigates how ATM networks are connected down to the layer 2 virtual circuit and port level. This agent also removes logical connectivity from LANE interfaces.
ILMIForeSys	The ILMIForeSys agent discovers physical ATM connections between devices by using the ILMI (Interim Local Management Interface) connectivity information provided by the Marconi ASX series of switches. When connectivity is deduced using ILMI information, it is usually the same as the connectivity that could have been calculated using PNNI information, as is the case with the standard AtmForumPnni and ILMI agents. However, there are some situations where the ILMI information contains details of a connection that is not in the PNNI information, and some situations where the PNNI information details a connection not in the ILMI information. The following examples detail situations where this may be the case: • Connections between ASX series switches and SE420/SE440 IADs are only discovered using ILMI. • Connections between Cisco routers or switches containing ATM cards and an ATM core are only discovered using ILMI. • As with the PnniForeSys agent, the ILMIForeSys agent is designed to operate seamlessly in conjunction with the ILMI agent. A network containing a mixture of ASX devices and another vendor's devices (for example, Cisco 5509 switches with ATM cards) can, therefore, be accurately discovered.
MariposaAtm	The MariposaAtm agent discovers the ATM connectivity of the SE420 and SE440 Integrated Access Devices (IADs). **Note:** The Ethernet switching and Frame Relay capabilities of these devices are not currently certified.

Table 139. ATM discovery agents (continued)

Agent name	Function
PnniForeSys	The PnniForeSys agent discovers physical ATM connections between devices by using the Private Network-to-Network Interface (PNNI) connectivity information provided by the Marconi ASX series switches. The PnniForeSys agent is designed to operate in conjunction with the AtmForumPnni agent. The PnniForeSys agent performs extra processing on Fore devices that do not store a logical ifIndex in their pnniLinkIfIndex variable. The information retrieved from these devices requires further processing to retrieve the actual ifIndex, which is held within the ifTable . **Note:** SNMP helper configuration for associated devices is a prerequisite for this agent. The AtmForumPnni agent must also be active.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

Discovering MPLS devices

To discover Multiprotocol Label Switching (MPLS) data, including Virtual Private LAN Service (VPLS) information, enable the appropriate agents.

The agents that retrieve MPLS data use either Telnet or SNMP to retrieve the data. Before enabling the MPLS agents, configure Telnet and SNMP access.

- Before enabling the MPLS agents that use Telnet, ensure that you have configured Telnet to enable the agents to access devices and to understand device output.
- Before enabling the MPLS agents that use SNMP, configure SNMP to enable access to devices and to specify threads, timeouts, and number of retries.

Tip: Agents that retrieve VPLS information can retrieve large amounts of data. Enabling these agents can add significant processing time to the discovery process. If you do not need to rediscover VPLS information, disable these agents for a faster discovery.

Table 140. MPLS discovery agents

Agent name	Function
CiscoMPLSSnmp	The CiscoMPLSSnmp agent discovers MPLS paths on Cisco devices using standard MIBs, and on Cisco devices that support the Cisco Experimental MPLS MIBs.
CiscoMPLSTelnet	The CiscoMPLSTelnet agent discovers MPLS paths and LDP VPLS on Cisco devices.
CiscoQinQTelnet	The CiscoQinQTelnet agent discovers QinQ (IEEE 802.1QinQ) conguration on Cisco devices.
HuaweiMPLSTelnet	The HuaweiMPLSTelnet agent discovers Layer 2 and Layer 3 MPLS/VPN related data on Huawei devices.

Table 140. MPLS discovery agents (continued)

Agent name	Function
JuniperMPLSTelnet	The JuniperMPLSTelnet agent discovers MPLS paths on Juniper devices. This agent also discovers Juniper MultiHome VPLS configurations and tags the Virtual Switch Instance (VSI) accordingly.
JuniperMPLSSNMP	The JuniperMPLSSNMP agent discovers MPLS/VPN (RT-based VPN discovery) and VPLS (LDP and BGP) related data on Juniper devices.
JuniperQinQTelnet	The JuniperQinQTelnet agent discovers QinQ (IEEE 802.1QinQ) conguration on Juniper devices.
LaurelMPLSTelnet	The LaurelMPLSTelnet agent discovers MPLS paths on Laurel devices. This agent is intended for route target-based discoveries only.
StandardMPLSTE	The StandardMPLSTE discovers MPLS Traffic Engineered (TE) tunnels using SNMP.
UnisphereMPLSTelnet	The UnisphereMPLSTelnet agent discovers MPLS paths on Juniper ERX routers (formerly Unisphere).

Multicast agents

Multicast agents retrieve data from devices participating in multicast groups and routes.

The agents that retrieve multicast data need SNMP and Ping access to retrieve the data. Before enabling the multicast agents, ensure that you first configured SNMP to enable the agents to access devices and to specify threads, timeouts, and number of retries.

The following table describes the multicast agents.

Table 141. Multicast discovery agents

Agent name	Function
StandardIGMP	Discovers networks running the Internet Group Management Protocol (IGMP). Supports any device that complies with the RFC2933 IGMP MIB. Depending on the level of MIB support, the following information may be discovered: IGMP Interfaces; Per-Interface Group Memberships; Group Members Visible on IGMP Interfaces.
StandardIPMRoute	Discovers IP multicasting networks. Supports any device that complies with the RFC2932 IPMRoute MIB. Depending on the level of MIB support, the following information may be discovered: Multicast Routing data (upstream/downstream); Interfaces involved in Multicast Routing; Multicast Sources and Groups.
StandardPIM	Discovers networks running the Multicast protocol PIM. Supports any device that complies with the RFC2934 PIM MIB. Depending on the level of MIB support, the following information may be discovered: PIM Interfaces; PIM Adjacencies; Candidate RPs/BSR.

Related tasks:

"Enabling the multicast agents" on page 34
To discover multicast groups, you must enable the appropriate agents and add the relevant SNMP community strings.

Discovering NAT gateways

There are several agents that download Network Address Translation (NAT) information from known NAT gateways.

None of the agents listed in the table below is enabled in the default configuration. These agents require advanced configuration, and it is preferable not to enable them by default.

Table 142. NAT gateway agents

Agent name	Function
CiscoNATTelnet	The CiscoNATTelnet agent interrogates Cisco routers acting as NAT gateways. This agent downloads the static NAT translations by means of TELNET from the device. The translations are then used to identify within which part of the network a particular device exists. **Note:** Before enabling this agent, it is necessary to configure Telnet access and the Telnet Helper.
NATNetScreen	The NATNetScreen agent interrogates NetScreen® Firewalls acting as NAT gateways. This agent downloads the static NAT translations by means of TELNET from the device. The translations are then used to identify within which part of the network a particular device exists. **Note:** Before enabling this agent, it is necessary to configure Telnet access and the Telnet Helper.
NATTextFileAgent	The NATTextFileAgent mimics the function of the other NAT gateway agents by reading NAT mapping information from a flat file. The translations are then used to identify within which part of the network a particular device exists. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

"DiscoTelnetHelperSchema.cfg configuration file" on page 71
The DiscoTelnetHelperSchema.cfg configuration file defines the operation of the Telnet helper, which returns the results of a Telnet operation into a specified device.

Discovering containment information

An important principle used by the network model is containment. A container holds other objects. You can put any object within a container and even mix different objects within the same container.

Containment information includes a physical breakdown of all parts held within the container, as well as detailed information on each of these parts. The parts that can be held within a container are:

- Chassis
- Interface
- Logical interface
- Vlan object
- Card
- PSU
- Logical collection, such as a VPN
- Module

There is also an Unknown category, which covers entities for which no part type has been defined.

The following table describes the discovery agents that discover containment information.

Table 143. Discovery agents that discover containment information

Agent name	**Function**
AvayaPhysicalInventory	The AvayaPhysicalInventory agent queries RAPID-CITY MIB for each physical entity and retrieves containment information for that physical entity. Run the AvayaPhysicalInventory agent if you want to model physical containment and perform asset management. Enable this agent if you have Avaya (formerly Nortel) devices in your network. **Note:** Configure SNMP access and the SNMP Helper before enabling this agent.
BNTSwitch	The BNTSwitch agent retrieves Layer 2 connectivity and VLAN containment information (including VLAN tags, VLAN Trunk, and Trunk Group information) using SNMP.

Table 143. Discovery agents that discover containment information (continued)

Agent name	**Function**
Entity	The Entity agent queries the MIB for each entity and retrieves containment information for that entity. Before enabling this agent, you must configure SNMP access and the SNMP Helper. Running the Entity agent during a discovery is optional. Some containment information is gathered during a discovery even if the Entity agent is not run. Run the Entity agent to model physical containment and perform asset management. **Note:** During a discovery, the Entity agent retrieves a large amount of data. This slows down the discovery. You should therefore only use this agent if you need to perform asset management on the retrieved data. You can configure the Entity agent to specify how much data the agent should retrieve. You can optionally choose to download this extra information from the entity MIBs of the , , Asset, ExtraPhysData, Module, Power, and Sensor entities. Do this by setting the following variables in the `Entity.agnt` file: • GetAssetData • GetExtraPhysData • GetModuleData • GetPowerData • GetSensorData In each case, set a value of `1` to retrieve the data, and set a value of `0` if you do not want to retrieve the data. The default value is `1`. In addition, you can specify how the Entity agent retrieves data from devices. The options are as follows: **`0` GetNext** This is the default value. Using this data retrieval option, the system requests one SNMP variable at a time from the device in series, that is, retrieval of one column in a table, one value at a time for a given device. This approach is slower but puts least pressure on the device. In a discovery with multiple entities the expectation is that overall this approach will not slow down the discovery as the SNMP helper is still busy with other activities. This approach might take a long time for individual large devices. This method works with SNMP version 1. **`1` Asynchronous GetNext** Similar to the GetNext method in that one index is retrieved at a time with the difference that all the columns are retrieved in parallel. This is also supported by SNMP version 1 and is faster but it also puts slightly more load on the device. **`2` GetBulk** Requests the entire column or multiple columns and individual Get commands in one go. This method requires SNMP version 2 support. If the device only supports version 1 then the retrieval method is broken down into multiple SNMP Get Next and Get commands. This is the fastest retrieval and it does not put much more load on the device than the Asynchronous GetNext method. This method also involves larger packets on the network. **Note:** The `Entity.agnt` file, together with all other agent configuration files, can be found in the `$NCHOME/precision/disco/agents` directory.
IfStackTable	The IfStackTable determines the interface stacking hierarchy on devices that support the RFC 2863 MIB. **Note:** Configure SNMP access and the SNMP Helper before enabling this agent.

Table 143. Discovery agents that discover containment information (continued)

Agent name	Function
JuniperBoxAnatomy	The JuniperBoxAnatomy agent retrieves information about which modules and components are installed in a Juniper device and their containment. The agent uses vendor-specific MIBs such as the Juniper Box Anatomy MIB.
JuniperERXIfStackTable	The JuniperERXIfStackTable determines the interface stacking hierarchy on Juniper ERX devices. This agent determines virtual-router and VRF context-sensitive stacking information for Juniper ERX devices. When a context-sensitive discovery is enabled this agent can be disabled, as the IfStackTable agent also determines this information. This will improve the performance of Discovery. **Note:** Configure SNMP access and the SNMP Helper before enabling this agent.
JuniperLAGStack	The JuniperLAGStack agent retrieves Link Aggregation Group (LAG) information from Juniper devices. LAG information is needed to accurately represent the interface stacking hierarchy.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

Discovery agents on other protocols

Network Manager provides agents that discover devices that use other protocols than ones previously described.

Note: Before enabling these agents, it is necessary to configure SNMP access and the SNMP Helper.

Table 144. Discovery agents on other protocols

Agent name	Function
AlteonStp	This is a Spanning Tree Protocol discovery agent for Alteon switches that support the dot1dStp section of the BRIDGE-MIB.
CDP	The CDP agent understands the protocol used among Cisco communication devices. Using CDP, Cisco devices can discover their nearest neighbors and store minimal information about them. This agent begins with the address of a known Cisco device and uses CDP to find more complete information about the locations of other connected or neighboring Cisco devices.

Table 144. Discovery agents on other protocols (continued)

Agent name	Function
DefaultLLDP	The DefaultLLDP agent discovers layer 2 connectivity between devices that support the LLDP MIB and have Link Layer Discovery Protocol (LLDP) enabled. Both the LLDP and DefaultLLDP agents use data from the LLDP MIB that is indexed by lldpRemLocalPortNum. This variable indicates which ifIndex or port a particular LLDP connection exists on. The LLDP agent supports devices where lldpRemLocalPortNum refers to the ifIndex on the device: typically, Cisco devices. The DefaultLLDP agent supports devices where lldpRemLocalPortNum refers to the port or other arbitrarily assigned index: typically, non-Cisco devices such as Juniper or BNT devices. The DefaultLLDP agent checks if the device supports the Extended-LLDP-MIB. If the device does not support the Extended-LLDP-MIB, lldpRemLocalPortNum is assumed to be a switch port. The agent then uses the dot1dBasePortIfIndex variable from the BRIDGE-MIB to determine the ifIndex of this record. Enable both the LLDP and DefaultLLDP agents so that Network Manager is able to find LLDP connectivity on devices that have different implementations of lldpRemLocalPortNum.
FddiDefault	The FddiDefault agent discovers any device that supports the standard FDDI MIB. When an FDDI device is interrogated, information relating to the interfaces of that device and its upstream and downstream neighbours is returned. The FddiLayer stitcher uses this and all other FDDI agents to determine the FDDI ring topology.
FddiCiscoConc	The FddiCiscoConc agent discovers Cisco Concentrator FDDI devices. Cisco concentrators know the full connectivity of every FDDI ring that passes though them, as opposed to just their upstream and downstream neighbours. Hence the FddiLayer stitcher gives the topology information returned by this agent precedence over that found by FddiDefault.
LLDP	The LLDP agent discovers layer 2 connectivity between devices that support the LLDP MIB and have Link Layer Discovery Protocol (LLDP) enabled. Both the LLDP and DefaultLLDP agents use data from the LLDP MIB that is indexed by lldpRemLocalPortNum. This variable indicates which ifIndex or port a particular LLDP connection exists on. The LLDP agent supports devices where lldpRemLocalPortNum refers to the ifIndex on the device: typically, Cisco devices. The DefaultLLDP agent supports devices where lldpRemLocalPortNum refers to the port or other arbitrarily assigned index: typically, non-Cisco devices such as Juniper or BNT devices. The LLDP agent checks if the device supports the Extended-LLDP-MIB. If it does, the agent retrieves the mapping between lldpRemLocalPortNum and ifIndex. If the device does not support the Extended-LLDP-MIB, lldpRemLocalPortNum is assumed to be the ifIndex. Enable both the LLDP and DefaultLLDP agents so that Network Manager is able to find LLDP connectivity on devices that have different implementations of lldpRemLocalPortNum.
SONMP	The SONMP agent uses the SynOptics Network Management Protocol, the protocol used between Nortel communications devices. The SONMP agent begins with the address of a known Nortel device and uses SONMP to discover location, containment, address, and connection information from connected, or neighbouring, Nortel devices.

Table 144. Discovery agents on other protocols (continued)

Agent name	Function
StandardSTP	The StandardSTP agent discovers STP connectivity data on any STP-enabled switch that supports the dot1dSTP section of the BRIDGE-MIB. You should run this agent in addition to any other necessary switch agents in order to discover STP backup (blocking) connections. The STP switch discovery method has the following advantages over other switch-based discovery methods: • Hidden links: STP backup (blocking) connections are discovered. • Speed: the agent completes in Phase 1; no pinging is required. **Note** : The STP agent only shows connections between STP enabled switches, that is, it ignores connections to nodes, non-switch devices, and non-STP enabled switches. This agent will not discover multiple STP instances, VLANs, or Virtual Routers.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

Context-sensitive discovery agents

There are several agents that take part in a context-sensitive discovery.

Attention: When a context-sensitive discovery is enabled, the discovery process automatically chooses the correct Context agent for any particular device. For this reason, you should not manually enable or disable Context agents, either through the configuration files or through the Discovery Configuration GUI.

Note: These agents require Telnet access and the Telnet Helper.

Table 145. Context-sensitive discovery agents

Agent Name	Function
RedbackContext	The RedbackContext agent discovers virtual router context-sensitive information for Redback® devices.

Table 145. Context-sensitive discovery agents (continued)

Agent Name	Function
UnisphereERXContext	The UnisphereERXContext agent discovers virtual router and VRF context-sensitive information for Juniper ERX devices. You can restrict the scope of the VRF contexts discovered by configuring the optional DiscoAgentDiscoveryScoping section in the .agnt file. The configurable options are: • IncludeVRF – allows the discovery of the named VRF. • ExcludeVRF – does not discover the specified VRF. VRF names are case-sensitive. The wildcard " * " can be used in place of a VRF name to apply the filter to all VRFs. If no filters are specified, all VRFs will be discovered by default.

Related concepts:

"Context-sensitive discovery" on page 8
If you need to discover devices such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

Related tasks:

"Configuring a context-sensitive discovery" on page 102
If you have devices that you need to discover such as SMS devices, MPLS Edge devices, or other devices with virtual routers, you must run a context-sensitive discovery. Context-sensitive discovery ensures correct representation of virtual routers. Always check that your particular device type is supported for discovery.

Related reference:

"DiscoConfig.cfg configuration file" on page 63
The DiscoConfig.cfg configuration file is used to have the Ping finder automatically check the devices discovered by the File finder, and to enable a context-sensitive discovery.

"TelnetStackPasswords.cfg configuration file" on page 78
The TelnetStackPasswords.cfg configuration file defines access credentials for Telnet access to devices.

"DiscoTelnetHelperSchema.cfg configuration file" on page 71
The DiscoTelnetHelperSchema.cfg configuration file defines the operation of the Telnet helper, which returns the results of a Telnet operation into a specified device.

Task-specific discovery agents

There is a group of discovery agents that are task-specific.

Table 146. Task-specific discovery agents

Agent name	Function
AlliedTelesynATSwitch	The AlliedTelesynATSwitch agent discovers Ethernet switches made by Allied Telesyn. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.

Table 146. Task-specific discovery agents (continued)

Agent name	Function
AlteonSwitch	The AlteonSwitch agent retrieves layer 2 connectivity information from Alteon load balancers and Ethernet switch modules. **Note:** Configure SNMP access and the SNMP Helper before enabling this agent.
ARPCache	The ARPCache agent assists in populating the Helper Server with IP to MAC address mappings in preparation for the Ethernet-based discovery agents. You must run this agent if you are running a layer 2 discovery. This agent is optional if you are running a layer 3 discovery. However, it can be more efficient to use the ARP Cache discovery agent because in most network environments the ARP helper can only run on one subnet at a time. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.
ASM	Determines whether ASMs for the following commercial server and database products are running on a device: • Oracle • Apache • Microsoft SQL Server • Microsoft Exchange • Microsoft Internet Information Server (IIS) • Microsoft Active Directory • IBM WebSphere® • BEA WebLogic • SAP • Sybase ASE • IBM Lotus® Notes/Domino Server The ASM agent determines whether an application is running by querying ASM-specific MIBs for the device. These MIBs are installed by default when you install Network Manager. The ASM agent can only retrieve this information from network devices on which ASM is deployed. Typically, you would deploy a ASM sub-agent on each commercial server and database product which is running on a device and whose performance you wish to monitor.

Table 146. Task-specific discovery agents (continued)

Agent name	Function
BGPPeerNextHop Interface	All PE to CE interfaces are added to a members list and an event on any of the interfaces in this members list causes the system to generate a synthetic MPLS VPN SAE. This agent, which is off by default, enables the generation of MPLS VPN service-affected events (SAEs) based on interfaces dependencies deeper in the core network. This agent calls the AddLayer3VPNInterfaceDependency.stch stitcher. This stitcher determines all PE to core provider router (P) interfaces and P to PE interfaces involved in a VPN. These PE -> P and P ->PE interfaces are added to a dependency list. An event on any of the interfaces in this dependency list causes the system to generate a synthetic MPLS VPN SAE. If an MPLS VPN SAE has already been generated based on an event on any of the interfaces in the members list, then any events in interfaces in the dependency list will be added as related events to that already generated MPLS VPN SAE.
CM	Retrieves data from cable modems that are connected to a cable modem termination system device. **Note**: If activated, this agent retrieves a large amount of information. Activating this agent may therefore place a heavy load on memory. You should only activate this agent if specific cable modem information is required beyond that provided by other agents.
CMTS	Discovers cable modem termination system devices. This agent also discovers cable modem connectivity. **Note**: If activated, this agent retrieves a large amount of information. Activating this agent may therefore place a heavy load on memory. You should only activate this agent if specific cable modem information is required beyond that provided by other agents.
ExtraDetails	The ExtraDetails agent is a text-based agent that builds on the basic SNMP information already retrieved by the Details agent.This agent retrieves the following information: • sysDescr • sysLocation • sysUpTime • sysServices • ifNumber **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.
HPNetworkTeaming	The HPNetworkTeaming agent discovers secondary NICs on HP Proliant Teamed network cards.If this agent is not enabled, only the primary NIC on an HP Proliant device will be discovered (as a local neighbour to the server) because only this NIC resides in the `ifTable`. This agent will create all NICs as local neighbours to the server. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.

Table 146. Task-specific discovery agents (continued)

Agent name	**Function**
LoopbackDetails	The LoopbackDetails agent is used to ensure that the management interface of a device is used in the topology and in subsequent monitoring as the main IP/name combination. The agent retrieves information needed to identify the management interfaces. This data is then used in the NamingFromLoopbackDetails stitcher. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.
MACFromArpCache	The ArpCache agent must be enabled for this agent to run. The MACFromArpCache agent is optionally activated in phase 3 of Discovery. It uses the ArpCache information retrieved by the ArpCache agent to retrieve the MAC address of the device. The agent is useful as it does not require SNMP access to the device to obtain the MAC address.
NetScreenArpCache	The NetScreenArpCache agent retrieves information from ARP tables configured in Netscreen devices and processes the tables to obtain the IP to MAC translation. The agent then sends the ARP information to the ARP Helper. After further processing, the ARP Helper sends the IP and MAC address mapping to the ARPHelperTable. The NetScreenArpCache agent uses the SNMPv2-SMI Standard MIB. **Note:** The ArpCache agent does not process the Netscreen devices processed by the NetScreenArpCache agent. This is to avoid conflict in the ipForwarding value as Netscreen is recognized as a non-routing device by the ArpCache agent.

Table 146. Task-specific discovery agents (continued)

Agent name	Function
NMAPScan	The NMAPScan agent is a Perl agent that runs the NMAP scanner against devices discovered by Network Manager. By default, the agent runs against devices that do not have SNMP access, or devices that have SNMP access but return sysObjectIds of devices from Apple, Compaq, IBM, Microsoft, Sun, Network Harmoni, UC David, Net-SNMP, and HP. The agent retrieves the following data: • Operating System Fingerprint details • TCP/UDP port and application information including port number, name, state, type, and service You must install NMAP version 4.85 or later on the same server where the Network Manager core components are installed. You must then edit the `NMAPScan.pl` file and specify the path to the NMAP binary in the **`my $nmapBinary`** line, and remove the comment from the beginning of the line. NMAP is available at http://nmap.org. **Attention:** Enabling the NMAPScan agent can extend the duration of the discovery. NMAP has a large number of scan options, refer to the NMAP documentation for more information. The following options are set by default for NMAP: • -sS: Perform a TCP SYN scan • -sV: Enable service version identification • -PN: Do not ping each target (Network Manager already uses the ping or file finder, or both) • -O: Enable Operating System fingerprinting • -oX: Enable XML output **Important:** Do not change this value.
OSInfo	Retrieves information about the operating system running on discovered devices. This agent only runs against Cisco and Juniper devices. The agent retrieves the following information: • OSType • OSVersion • OSImage
SSM	The SSM agent retrieves MIB information by SNMP from devices running SSM agents. This agent retrieves information such as the software installed on the device, running processes, CPU utilization, storage devices on this entity, free disk space, and so on. The SSM agent can only retrieve this information from network devices on which the SSM Agent is deployed. Typically, you would deploy a SSM Agent on devices whose performance you wish to monitor. For more information on the SSM Agent, see the *SSM Application Guide*. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.

Table 146. Task-specific discovery agents (continued)

Agent name	Function
SSMOracle	The SSM application and the Oracle monitoring package must also be running. The SSMOracle agent retrieves MIB information by SNMP from devices running SSM agents. This agent retrieves information such as the Oracle database names, fields, and database sizes. The SSMOracle agent can only retrieve this information from network devices on which the SSM Agent is deployed. Typically, you would deploy a SSM Agent on devices whose performance you wish to monitor. For more information on the SSM Agent, see the *SSM Application Guide*. **Note:** Before enabling this agent, it is necessary to configure SNMP access and the SNMP Helper.
TunnelAgent	Template for a Perl agent to retrieve information about all tunnels, including IPv6 over IPv4 tunnels, present in the network. This agent works in conjunction with the IPv6Interface agent.

Related reference:

"SnmpStackSecurityInfo.cfg configuration file" on page 75
The SnmpStackSecurityInfo.cfg configuration file defines the community strings, versioning, and other properties used by any process that needs to interrogate devices using SNMP, for example, the SNMP helper. Community strings can be configured on a per-device or per-subnet basis, to allow the SNMP Helper to retrieve MIB variables from devices.

"DiscoSnmpHelperSchema.cfg configuration file" on page 70
The DiscoSnmpHelperSchema.cfg configuration file defines the operation of the SNMP Helper, which specifies the general rules of SNMP information retrieval.

Discovery agents for IPv6

Network Manager provides a Perl agent template that you can use as a base for developing Perl agents to retrieve IPv6 interface data.

Table 147 describes the Perl agent templates.

Note: Rather than having agents with specific IPv6 capabilities, most of the discovery agents have IPv6 capabilities; for example, the InetRouting agent supports IPv6 routing entries but it also downloads IPv4 interfaces and route information.

Table 147. IPv6 agent template

Agent name	Function
IPv6Interface	Template for a Perl agent to retrieve interface information from an IPv6 device. This agent is designed to work in an identical way to the Interface agent. This agent template is located in the Perl agents directory at the following location: `$NCHOME/precision/disco/agents/perlAgents`.

Guidance for selecting agents

To discover device technologies (that is, those that use protocols other than IP) on your network, you must ensure that the appropriate agents are active.

The following list provides the non-IP device protocols that are supported by Network Manager. You can select the appropriate agents for these protocols.

- Frame Relay
- Private Network-Network Interface (PNNI)
- Cisco Discovery Protocol (CDP)
- Link Layer Discovery Protocol (LLDP)
- Hot Standby Routing Protocol (HSRP)
- Fibre Distributed Data Interface (FDDI)
- Asynchronous Transfer Mode (ATM)
- Integrated Local Management Interface (ILMI)
- Multiprotocol Label Switching (MPLS)

Which IP layer agents to use

The IP layer agents that you need to use depend on the devices on your network:

- If you do not want to have your IP routing tables accessed, you must only use the IpBackupRoutes agent.

  This agent is not used by default as it has the following drawbacks:
  - It retrieves data from a table that is not dynamic. If the router has not been refreshed, then the data retrieved by this agent may be spurious.
  - The table is large and therefore takes a long time to download.
- If there are modern devices on the network, you must use the IpRoutingTable agent and the IpForwardingTable agent.

  These agents provide an accurate picture of IP layer connectivity and are therefore used by default.

Which standard agents to use

The standard agents that you need to use depend on the information you require and the devices on your network.

- The TraceRoute agent can be used if there is a firewall on the network, because SNMP calls cannot always be made through firewalls. If you use the TraceRoute agent, you must specify, as a discovery seed, the subnet node for the subnet on the other side of the firewall.
- The ArpCache agent retrieves the physical address of a device, so is only required (in conjunction with the Switch agents) when performing layer 2 discoveries.
- Frame Relay agents should be run in conjunction with the IP layer agents if you need to add DLCI information to the interfaces of Frame Relay devices.
- Switch agents must be run for a layer 2 discovery.
- The device-specific and protocol-specific agents are only required to discover the devices or protocols to which they relate.

Which specialized agents to run

Several agents need to run only when you need to discover certain device types or network technologies.

The specialized agents that you need to run depend on the devices and protocols in your network:

- The Extreme agent can be used to extract layer 2 connectivity information, EDP neighbors, and VLAN details from Extreme switches.
- The ExtremeESRP agent discovers Extreme Standby Routing Table information from Extreme routing switches.
- The PnniForeSys agent discovers physical ATM connections between devices by using the PNNI (Private Network-to-Network Interface) connectivity information provided by the Marconi ASX series switches.
- The ILMIForeSys agent discovers physical ATM connections between devices by using the ILMI (Interim Local Management Interface) connectivity information provided by the Marconi ASX series switches.
- The CellPath90 agent discovers the ATM connection of a CellPath 90 WAN (Wide Area Network) multiplexer.
- The Marconi3810 agent discovers the Ethernet connectivity of the ES-3810 switches running operating system version 4.x.x.
- The MariposaAtm agent discovers the ATM connectivity of the SE420 and SE440 IADs.

  Note: The Ethernet switching and Frame Relay capabilities of these devices are not currently certified.
- The ILMI agent discovers connectivity between ATM devices running ILMI that support the ATM Forum's ATM MIB. The CiscoPVC agent retrieves PVC data from Cisco devices.
- The AtmForumPnni agent discovers connectivity between devices running ATM Forum PNNI that correctly support the ATM Forum's PNNI MIB.
- For Cisco devices, run the CiscoMPLSSnmp agent if you have the MPLS MIBs enabled on a device, otherwise, use the CiscoMPLSTelnet agent.
- For Juniper devices, run the JuniperMPLSTelnet agent if you wish to discover MPLS paths.
- For Juniper ERX devices (formerly Unisphere), the UnisphereMPLSTelnet agent must be used to discover MPLS paths, as these devices are sufficiently different to the Juniper "M" series routers that a different agent is required.
- The StandardMPLSTE agent discovers MPLS Traffic Engineered (TE) tunnels.
- The StandardIGMP agent discovers networks running the Internet Group Management Protocol (IGMP).
- The StandardIPMRoute agent discovers IP multicasting networks.
- The StandardPIM agent discovers Protocol Independent Multicast (PIM) groups.

Related reference:

"Types of agents" on page 311
The agents supplied with Network Manager can be divided into categories according to the type of data they retrieve or the technology they discover.

Suggested agents for a layer 3 discovery

The recommended agents for a layer 3 discovery depends on your network.

When running a layer 3 discovery, the following agents should be run:

- Details and AssocAddress
- A combination of the following IP layer agents:
  - IpRoutingTable
  - IpBackupRoutes
  - IpRoutingTable and IpForwardingTable
- HSRP
- VRRP
- TraceRoute (if firewalls are present)
- IPv4/6 InetRouting. If you have IPv6 in your network, consider running this agent to discover the connectivity, particularly the IPv6 connectivity.

Tip: Some routers support layer 2 technologies. For example, when an ATM card is located in a router chassis, layer 3 discovery agents, such as the IpRoutingTable agent, only discover interfaces with an IP address. Therefore, to fully discover all the interfaces on routers that support layer 2 technologies, you must run the appropriate agents.

Related reference:

"Suggested agents for a layer 2 discovery"
The recommended agents for a layer 2 discovery depends on your network.

Suggested agents for a layer 2 discovery

The recommended agents for a layer 2 discovery depends on your network.

When running a layer 2 discovery, the following agents must be run:

- Details and AssocAddress
- A combination of the following IP layer agents:
  - IpRoutingTable
  - IpBackupRoutes
  - IpRoutingTable and IpForwardingTable
- Switch
- FrameRelay
- ArpCache
- ATM
- FDDI
- HSRP
- VRRP
- MPLS

Related reference:

"Suggested agents for a layer 3 discovery"
The recommended agents for a layer 3 discovery depends on your network.

Appendix D. Helper System

The helpers are specialized applications that retrieve information from the network on demand.

Note: If the helpers and the Helper Server are running on a different host to the DISCO process, and these hosts are behind a firewall, then specialized configuration is required to ensure that the Helper System can communicate with DISCO. For more information, see the *IBM Tivoli Network Manager IP Edition Administration Guide*.

Helpers

Helpers retrieve information from devices and deposit the information in the Helper Server for retrieval by the agents.

By default, there are six helpers, which are described in Table 148.

Table 148. Helpers available with Network Manager.
Note: $NCHOME is the environment variable that contains the path to the netcool directory.

Helper	Executable	Configuration file	Description
ARP	ncp_dh_arp	$NCHOME/etc/precision/DiscoARPHelperSchema.cfg	Performs IP address to MAC address resolution.
DNS	ncp_dh_dns	$NCHOME/etc/precision/DiscoDNSHelperSchema.cfg	Performs IP address to device name resolution.
PING	ncp_dh_ping	$NCHOME/etc/precision/DiscoPingHelperSchema.cfg	Either pings each device in a subnet, an individual IP address or a broadcast or multicast address. The result of the ping could be used to populate the MIB of the device.

SNMP	ncp_dh_snmp	`$NCHOME/etc/precision/DiscoSnmpHelperSchema.cfg`
		`$NCHOME/etc/precision/SnmpStackSchema.cfg`
		`$NCHOME/etc/precision/SnmpStackSecurityInfo.cfg`
			Returns results of an SNMP request such as Get, GetNext and GetBulk.

TELNET	ncp_dh_telnet	`$NCHOME/etc/precision/DiscoTelnetHelperSchema.cfg`
		`$NCHOME/etc/precision/TelnetStackPasswords.cfg`
		`$NCHOME/etc/precision/TelnetStackSchema.cfg`
			Returns the results of an OS command against a specific device using the Telnet or SSH protocol.

Table 148. Helpers available with Network Manager (continued).

Note: $NCHOME is the environment variable that contains the path to the netcool directory.

Helper	Executable	Configuration file	Description
XMLRPC	ncp_dh_xmlrpc	$NCHOME/etc/precision/ DiscoXmlRpcHelperSchema.cfg	Enables Network Manager to communicate with EMS collectors using the XML-RPC interface.

Helper System operation

At startup, the Helper Server loads up the Helper Server schema from the DiscoHelperServerSchema.cfg configuration file and creates the appropriate helper databases. The Helper Server also creates a Helper Manager for every helper database.

The Helper Manager manages the way in which the helper handles requests from the Helper Server to retrieve network device data. The Helper Manager specifies:

- The request timeout
- The time-to-live for the returned variables
- Whether multiple requests are to be processed in serial or parallel

When the Helper Manager detects a request for network data from the Helper Server, it instructs the associated helper to retrieve the data from the network.

Dynamic timeouts

The Helper System uses dynamic timeouts to handle network requests.

As an example of the benefit of dynamic timeouts, if the SNMP helper is asked to perform numerous SNMP Get requests, the helper might begin to slow down and therefore exceed the timeout. A static timeout would cause the retrieval of data to terminate (with data lost) even though the device is still responding with data.

To prevent this situation, the helpers incorporate a dynamic timeout system in which they note SNMP Get requests and recalculate and update the timeout as the SNMP daemons of the device begin to slow down.

Appendix E. Discovery stitchers

Stitchers are processes that transfer, manipulate, and distribute data between databases. The discovery stitchers also process the information collected by the agents and using this information to create the network topology.

The discovery stitchers supplied with Network Manager are stored in the following directories.

- Text-based discovery stitchers (text files with a .stch extension): `$NCHOME/precision/disco/stitchers/`
- Precompiled discovery stitchers : `$NCHOME/precision/platform/`*`platform`*`/lib/`, where *platform* is the operating system on which Network Manager is running; for example, `linux2x86`, `win32`, `solaris2`, `aix5`, `hpux11`, or `linux2s390`.

For information on stitcher language, see the *IBM Tivoli Network Manager IP Edition Language Reference*.

Main discovery stitchers

This topic lists all discovery stitchers.

The following table describes the discovery stitchers currently included with Network Manager.

Note: This list is subject to change.

Table 149. List of Network Manager discovery stitchers

Stitcher	Function
AddAEPhysicalIFContainment	Adds physical interfaces to the chassis in the Link Aggregation Group (LAG) containment structure of Juniper devices. This stitcher is called by BuildContainment.stch.
AddBaseNATTags	Updates all the private NAT addresses that have a private address with their public address and adds a tag denoting the private address.
AddBasicContainment	Part of the mechanism for containment stitching. This stitcher inserts containment information into the simple chassis.
AddCardContainment	Adds card objects to the `workingEntities.containment` table.
AddContainedByAttribute	Adds an ExtraInfo attribute called m_PhysicallyContainedBy. It is analogous to the RFC2737 attribute entPhysicalContainedIn and identifies the record that contains a particular record. This data is used by Netcool® for Asset Management and the Cramer integration and must be uncommented in the PostScratchProcessing stitcher when using those applications.
AddEntityContainment	Adds general entity information to the `workingEntities.containment` table.
AddGlobalVlans	Builds global Virtual Local Area Network (VLAN) objects using the `translations.vlans` table.
AddIfStackContainment	Adds interface stack objects to the `workingEntities.containment` table.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
AddJuniperEntityContainment	Adds containment information for interface port entities to the device records of Juniper devices

AddLayer3VPNInterfaceDependency	This stitcher determines all PE to core provider router (P) interfaces and P to PE interfaces involved in a VPN. These PE -> P and P ->PE interfaces are added to a dependency list. An event on any of the interfaces in this dependency list causes the system to generate a synthetic MPLS VPN SAE. If an MPLS VPN SAE has already been generated based on an event on any of the interfaces in the members list, then any events in interfaces in the dependency list will be added as related events to that already generated MPLS VPN SAE.
	The BGP sessions set up between the PE speakers, and consequently, the VPNs, depend on the PE -> P and P -> PE interfaces for a given VPN and PE pair. The value of adding these interfaces to the VPN dependency list is that it allows the P->PE and PE->P links to be considered in Service Affected Event (SAE) calculations and thus provide a notification that some set of VPNs on a PE are affected by a link problem between PE and P routers.
	The diagram below marks with an asterisk the interfaces that the AddLayer3VPNInterfaceDependency stitcher adds as an MPLS VPN SAE dependency. In this diagram, the following conventions are used:
	• `[ce]` is a customer-edge router
	• `[PE` is a provider-edge router
	• `[P]` is a provide core router
	`[ce]---[PE]*---*[P]---[P]---[P]*---*[PE]---[ce]`
	`                       |*`
	`                       |`
	`                       |*`
	`                      [PE]---[ce]`
	The results of the stitcher manifest themselves as the m_DependsOn list in the following sample record which shows that an example VPN, VPN_CONTAINER_ACME consists of a number of interfaces in the VPN (m_Members list contains the PE->CE facing interfaces) and subsequently depends on the PE->P/P->PE facing interfaces in the m_DependsOn list.
	`{`
	`m_Name='VPN_CONTAINER_ACME';`
	`m_Creator='STITCHER CREATED';`
	`m_Description='Logical object for VPN ACME';`
	`m_EntityType=7;`
	`m_ObjectId='VIRTUAL_PRIVATE_NETWORK';`
	`m_HaveAccess=0;`
	`m_IsActive=0;`
	`m_ExtraInfo={`
	`     m_VPNName='ACME';`
	`     m_MPLSVPNType='MPLS IP VPN MESH';`
	`     m_Members=['pe7-cr38.core.eu.test.lab[Vl2]',`
	`       'pe7-cr38.core.eu.test.lab[Fa0/3/1]',`
	`       'pe8-cr72.core.eu.test.lab[Fa5/0]'];`
	`     m_DependsOn=['pe7-cr38.core.eu.test.lab[Se0/0/0:0.202]',`
	`       'pe8-cr72.core.eu.test.lab[Fa0/0]',`
	`       'p4-cr28.core.eu.test.lab[Se0/0/1:0.202]',`
	`       'p4-cr28.core.eu.test.lab[Gi0/0]'];`
	`             };`
	`}`

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
AddLogicalToIpToBaseName	Adds logical information to the translations.ipToBaseName table.
AddLoopbackTag	Adds a tag to the ExtraInfo column of the topology database indicating that an interface is a globally addressable loopback interface.
AddNoConnectionsToLayer	The final topology layer is constructed by merging the topology information from the various layers. If there is a mismatch in connectivity information provided by the different layers, information from the more detailed layer takes precedence. For example, the Network Layer (layer 3) provides information indicating that a router interface is connected to another router interface. However, information from the more detailed Data Link Layer (layer 2) shows that there is actually a switch between the two router interfaces. The AddNoConnectionsToLayer is used in cases where it is necessary to remove a connection at one layer but keep the connection at a different layer.
AddOSPFAreaCollections	Creates a logical collection for each OSPF area containing the interfaces within that area.
AddSwitchRoutingLinks	Adds switch routing data (that assists the RCA plug-in when performing Root Cause Analysis) to the topology database.
AddTechnologyType	Optional stitcher called by the PostScratchProcessing.stch stitcher. This stitcher is commented out by default. If enabled, this stitcher creates a technology type variable for each interface object. This variable can then be used to create technology-based Network Views. See the *IBM Tivoli Network Manager IP Edition Network Visualization Setup Guide* for more information about network views. The stitcher creates the technology type variable by adding an m_Technology field to the ExtraInfo field within the scratchTopology.entityByName table for each interface object. The m_Technology field is a string, such as Ethernet, ATM. The stitcher contains a large collection of default technology types; more can be added by directly altering the stitcher. The small processing load associated with activating this stitcher might slow down your discovery slightly.
AddUnconnectedContainment	Gives unconnected entities a default containment. Unconnected entities do not have a parent, except for their main node or interface.
AddUnmanagedHub	Infers the existence of an unmanaged hub if the discovery finds a port that is connected to more than one item. This stitcher then connects that port to an unmanaged hub, and also connects all the other ports involved in the connection to the unmanaged hub. These connections make the topology clearer.
AddVlanContainers	Uses information in the workingEntities.finalEntity and translations.vlans tables to add VLAN objects to the workingEntities.containment table.
AddVTPCollections	Augments the VTP domain entities with ports that are connected to VTP domains.
AddVTPEdges	Augments the VTP domain entities with ports that are connected to VTP domains.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
AdjustedIPLayer	Adjusts the IP layer to move the IP layer connectivity on logical interfaces down to the physical interface for some routers.
AgentRetProcessing	Processes data from the `returns` table of each table.
AgentRetToInstrumentationCiscoFrameRelay	Populates the `instrumentation.ciscoFrameRelay` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationFddi	Populates the `instrumentation.fddi` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationFrameRelay	Populates the `instrumentation.frameRelay` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationHSRP	Populates the `instrumentation.hsrp` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationIp	Populates the `instrumentation.ip` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationName	Populates the `instrumentation.name` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationPnniPgi	Populates the `instrumentation.pnniPeerGroup` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationSubnet	Populates the `instrumentation.subNet` table with information from the `returns` table of the appropriate agent.
AgentRetToInstrumentationVlan	Populates the `instrumentation.vlan` table with information from the `returns` table of the appropriate agent.
AgentStatus	This stitcher sends events to the `disco.events` table about the status of the discovery agents. These events indicate changes in the state of the agent; for example, if it has started, has finished, or has crashed. See also, FinderStatus, CreateStchTimeEvent, and DiscoEventProcessing stitchers.
AnalyseTopology	Analyses a connectivity database to find how many connections there are on each interface.
AnalyseTopologySummary	This stitcher uses the analysis summary information produced by the AnalyseTopology stitcher to provide an optional deeper topology analysis. This functionality is kept separate from the basic topology analysis as it might affect performance or create topology issues on some networks.
AnalyseTopology	Analyses a connectivity database to find how many connections there are on each interface.
AnalyseTopologySummary	This stitcher uses the analysis summary information produced by the AnalyseTopology stitcher to provide an optional deeper topology analysis. This functionality is kept separate from the basic topology analysis as it might affect performance or create topology issues on some networks.
ApplyMainDisplayLabel	Sets the display label for devices in the GUI based on the setting of m_DisplayMode in the disco.config configuration file. Modifies the entities in the workingEntities.finalEntity database table. Called by the BuildFinalEntityTable.stch and RebuildFinalEntityTable.stch stitchers.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
ASMAgentRetProcessing	Based on MIB variable data retrieved by the ASM stitcher, this stitcher generates a list of ASM sub agents running on a given device. Each ASM subagent running on a device corresponds to a commercial server or database product running on that device. The list of ASMs enables autopartitioning of devices within a network based on the commercial server or database products running on those devices.
ASAMIfStringLayer	Uses the ASAM ifDescr format to deduce connectivity.
ASMProcessing	Updates entities based on the services running on them.
ASRetProcessing	Used in MPLS discoveries where devices in different customer VPNs have identical IP addresses. This stitcher performs the processing necessary to differentiate between these devices and correctly resolve device connectivity. This stitcher is called by the AsAgent agent and works with the `ASMap.txt` file in `NCHOME/precision/etc`.
AssocAddressRetProcessing	Processes data in the `AssocAddress.returns` table, sending the device details to the appropriate discovery agent if the device has not already been discovered.
BGPLayer	Builds the BGP layer created by BGP agent. In common with other layer stitchers, this stitcher receives input from relevant agents. This input consists of entity records containing local and remote neighbor data fields. The stitcher uses these records to work out the local and remote connections for each entity.
BuildBaseSubnetRegex	Takes a given subnet and mask and produces a regular expression to find IP addresses in that subnet.
BuildContainment	Calls the following stitchers to add different types of objects to the `workingEntities.finalEntity` table: • `AddBasicContainment` stitcher, which adds device containment information. • `AddCardContainment` stitcher, which adds card containment information. • `AddIfStackContainment` stitcher, which adds interface stack containment information. • `AddEntityContainment` stitcher, which adds general containment information. • `NATAddressSpaceContainment` stitcher, which adds containment information associated with NAT address spaces. • `AddVlanContainers` stitcher, which adds VLAN containment information. You can comment out lines in this stitcher as appropriate in order to exclude types of objects that are not needed. **Note:** This stitcher also manages collector-discovered devices by accepting data from the CollectorInventory agent.
BuildFinalEntity	Builds the records for a single chassis. The BuildFinalEntity stitcher merges data from multiple agents to create the complete definition of an entity. This stitcher is called by the `BuildFinalEntityTable` stitcher.
BuildFinalEntityTable	Uses the entries in the `translations.ipToBaseName` table to populate the `workingEntities.finalEntity` table.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function

BuildInterfaceName	Used to control the naming of interfaces. By default, this stitcher is called by the `BuildFinalEntity` stitcher.
	The default naming strategy for any device interface is as follows:
	`baseName[<card>[<port>]]`
	Alternatively Network Manager uses the following default naming convention if the card and port are not valid:
	`baseName[0[<ifIndex>]]`
	You can use the `BuildInterfaceName` stitcher to change the naming convention for an interface in one of the following ways:
	• Specify that you want to use `ifName` or `ifDescr` to name the interfaces rather than the `ifIndex`, card or port information. Using this option, interfaces would have names like the following example:
	`baseName[eth0/0]`
	In this example `eth0` is the `ifName` of an interface.
	To change the naming convention in this way, change the value of `m_UseIfName` in the `disco.config` table.
	• Modify the `BuildInterfaceName` stitcher directly to specify any interface naming convention.

BuildLayers	Activated in the final phase to implement the stitchers that build the layer databases.
BuildMPLSContainers	This stitcher calls the BuildVPNContainers and BuildVRAndVRFContainers stitchers. It builds VPN, VR, and VRF containers.
BuildNATTranslation	Builds a global translation table for all NAT devices.
BuildVPNContainers	Creates objects to represent the MPLS VPNs within the system.
BuildVRAndVRFContainers	Creates virtual router (VR) and virtual routing and forwarding table (VRF) objects within the system. These objects are useful for displaying MPLS information.
BuildVSIContainers	Creates Virtual Switch Instance (VSI) and Virtual Forwarding Instance (VFI) entities. This stitcher also creates logical containment of devices associated with VSIs, VFIs, and CE-PE links.
CabletronLayer	Determines connectivity information based on Cabletron data returned by the discovery agents.
CDPLayer	Determines connectivity information based on the data returned by the CDP agent.
CheckAndSendNATGatewaysToArpCache	Sends the NAT gateways to the ArpCache agent.
CheckForMasterLink	Looks for connections lower down the interface stack that take precedence over connections higher up the stack.
CheckIfMgmtAddress	Determines if a given IP address is a defined management address.
CheckIndirectResponse	Handles indirect ICMP responses due to NAT.
CheckInterfaceStatus	Checks the `ifOperStatus` data and updates the interfaces status where the `ifOperStatus` is not `1`.
CheckManagedProcesses	Checks if the processes in disco.managedProcesses have been started, and if they have not been started it attempts to start them.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
CheckMultipleIPNoAccess	Checks for devices with no access but multiple IP addresses. Creates interface objects for these IP addresses and updates the entity appropriately.
CheckValidVirtual	Determines if the given IP address is a valid virtual IP address.
CiscoSerialInterfaceLayer	Creates a new layer called CiscoSerialInterfaceLayer connecting Cisco switches that are connected by serial interfaces. By default, the stitcher removes any connections in the CiscoSerialInterfaceLayer that are duplicated in the IPLayer database, to prevent wrong connectivity. The function to remove mesh connections can be turned on or off by editing a flag in the stitcher.
CiscoVSSContainment	CiscoVSSContainment adds new containment entities, representing the two physical chassis and their respective interfaces and objects, to the workingEntities.finalEntity table.
CMTSLayer	Uses the data downloaded by the CMTS agent to build the connection information between cable modem termination systems and the attached cable modem devices.
ContextAgentRetProcessing	This stitcher is used for context-sensitive discovery data flow. It merges the outputs of all the Context agents for each entity. It then inserts the results of this merge into the `AssocAddress.despatch` table, using the DetailsOrContextRetProcToAgent stitcher.
CollectorAddressTranslation	This stitcher processes devices discovered using an EMS collector. This stitcher performs the following activities: • Ensures that any collector-discovered devices are identified as being the same as the equivalent SNMP-discovered devices. • Stores data on the collector associated with each device. • Performs other administration tasks related to a collector discovery.
CollectorDetailsRetProcessing	This stitcher processes devices discovered using an EMS collector. It processes entries in the returns table of the CollectorDetails agent and sends these entries to other collector discovery agents. The collector discovery agents retrieve detailed device data from the EMS collectors.
CollectorIPLayer	This stitcher builds layer 2 connectivity for devices discovered using an EMS collector based on the connection data supplied by the CollectorLayer2 agent.
CollectorLagLayer	Creates EMS-based Layer 2 connectivity from Alcatel Lucent 5620 Collector Link Aggregation (LAG) information.
CollectorSwitchLayer	This stitcher builds layer 3 connectivity for devices discovered using an EMS collector based on the connection data supplied by the CollectorLayer3 agent.
CreateAndSendTopology	Activates the stitchers that create the topology and send the final Scratch Topology to MODEL.
CreateBGPAutonomousSystems	Creates and names a BGP autonomous system (AS). Provides the option to resolve the AS number to AS name, which enables display of a customer or business-related name when visualizing the AS in a topology map. Also retrieves data that indicates whether an AS is single-homed. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateBGPNetworksCollection	Creates a topology database record, known as a BGP network, that groups a collection of BGP autonomous systems. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
CreateBGPProtocolEndPoints	Creates BGP protocol endpoints. A BGP protocol endpoint is a logical interface that can be used by the BGP hosted service on a device. A physical port can implement multiple BGP protocol endpoints. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateBGPServices	Creates BGP hosted service entities. A hosted service is a service or application running on a specific device. For example, a device might host BGP and OSPF services. Each BGP hosted service entity describes a BGP process on a router. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateBGPTopology	Creates connections between BGP speakers. These connections are presented in the Network Views, and correspond to working BGP connections at the time of the discovery. This stitcher can also infer BGP peer routers that Network Manager cannot access. These inferred routers might correspond to BGP autonomous systems outside of your company. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateIGMPGroups	Creates multicast group entities and adds associated IGMP end points as members. The Group entities populate the igmpGroup NCIM table.
CreateIGMPProtocolEndPoints	Creates Multicast IGMP Protocol End Point entities, which populate the igmpEndPoint NCIM table.
CreateIGMPServices	Creates Multicast IGMP Service entities, which populate the igmpService NCIM table.
CreateImpactTopology	An optional stitcher that can be used to make a copy of the Scratch Topology before it is sent to the Topology Manager, ncp_model.
CreateIPMRouteGroups	Creates the MDT, Group and Source entities that populate the ipMRouteMDT, ipMRouteGroup, and ipMRouteSource NCIM tables.
CreateIPMRouteProtocolEndPoints	Creates Multicast Routing Protocol Endpoint entities which populate the ipMRouteEndPoint NCIM table.
CreateIPMRouteRoutes	Manages the creation of upstream and downstream route entities for the routes downloaded from multicast routers. It also aids MDT resolution.
CreateIPMRouteTopology	Populates the IPMRoute adjacency fields that populate the IPMRoute topology in NCIM.
CreateIPMRouteServices	Creates Multicast Routing Service entities which ultimately populate the ipMRouteService NCIM table.
CreateMPLSTEResources	Creates MPLS TE Resource entities.
CreateMPLSTEServices.stch	Creates the MPLS Tunnel Engineering (TE) Service entities and associates them with their host chassis entities.
CreateMPLSTETunnels.stch	Creates the MPLS TE Tunnel entities and associates them with the appropriate TE Service entity.
CreateMPLSTEProtocolEndPoints.stch	Creates the MPLS TE Protocol End Points and associated then with appropriate TE Service entity.
CreateMPLSTENetworkPipes.stch	Creates Network Pipe fields in the Tunnel entities. The pipes consist of IP Connection entities that represent the path of the Tunnel.
CreateMPLSTEPipeHop.stch	Creates IP Connection entities for use with NetworkPipes.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
CreateMPLSTETopology.stch	Adds MPLS TE link fields to interface entities involved in TE Tunnel paths.
CreateMXGroupCollection	Creates collections based on the master IP address for a group of routing engines. Called by the PostScratchProcessing stitcher.
CreateNetworkManagementCards	This stitcher creates NetworkManagementCard objects.
CreateOSPFAreas	Creates and names an OSPF area. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateOSPFNetworkLSAPseudoNodes	Retrieves data associated with OSPF pseudonodes advertised by designated routers and builds these pseudonodes in the topology. This overcomes the problem of full meshing when representing OSPF area in Network Views and enables connections within OSPF areas to be visualized in a clear, uncluttered manner.
CreateOSPFPointToPointAdjacencies	Retrieves data associated with point-to-point connections within an OSPF area and creates these connections in the topology. These connections are shown in Network Views. Only enabled connections are shown.
CreateOSPFProtocolEndPoints	Creates OSPF protocol endpoints. An OSPF protocol endpoint is a logical interface that can be used by the OSPF hosted service on a router. This stitcher also gathers data that indicates which OSPF area an endpoint is in. A physical port can implement multiple OSPF protocol endpoints. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateOSPFRoutingDomains	Creates a topology database record, known as an OSPF routing domain, that groups a collection of OSPF areas. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreateOSPFServices	Creates OSPF hosted service entities. A hosted service is a service or application running on a specific device. For example, a device might host BGP and OSPF services. Each OSPF hosted service entity describes an OSPF process on a router, and also indicates whether the router on which the OSPF service is running is an area border router or an AS border router. This stitcher is called by the PostScratchProcessing stitcher following the creation of the scratch topology.
CreatePIMNetworksCollection	Creates a collection entity to collect PIM enabled routers.
CreatePIMProtocolEndPoints	Creates protocol endpoints for each PIM interface.
CreatePIMServices	Creates a device level service object representing the state of the hosted multicast service, and a link from the chassis to this service object.
CreatePIMTopology	Creates the PIM topology using PIM adjacency information rather than m_RouterLinks.
CreateScratchTopology	Creates the scratch topology.
CreateStchTimeEvent	This stitcher sends events to the `disco.events` table about progress within the data processing phase. For example, the stitcher generates events to indicate that the discovery process has started building the working entities table, and that the discovery process has started building the containment table. See also, AgentStatus, FinderStatus, and DiscoEventProcessing stitchers.
CreateTrunkConnections	Modifies the containment model to take account of VLAN trunks.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
CreateVlanEntity	This stitcher creates a single VLAN entity object by adding VLAN data to the Scratch Topology.
CreateVRRPCollection	Creates collections based on the Virtual Router Redundancy Protocol (VRRP) virtual router ID and associated IP address. Called by the PostScratchProcessing stitcher.
DetailsOrContextRetProcToAgent	This stitcher is used as part of the context-sensitive discovery data flow. It is equivalent to DetailsRetProcessing but handles context-sensitive discovery. It processes entities from the `details.returns` table, and sends the details to the despatch table of the relevant Context agent.
DetailsRetProcessing	Processes entities from the `details.returns` table, and sends the details to the `AssocAddress.despatch` table.
DetectionFilter	Determines whether a given device passes the detection filter and is to be discovered based on the `detectionFilter` defined in the `scope` database. By default, the discovery filters do not filter out the Network Manager server, because this server usually also serves as the polling station for root cause analysis. In order for root cause analysis to work correctly, the polling station, and hence the Network Manager server, must be part of the topology. If you need to filter out the Network Manager server using the `detectionFilter`, modify the DetectionFilter stitcher and remove the sections of code indicated by comments that prevent the Network Manager server from being filtered.
DiscoEventProcessing	This stitcher responds to an insert into the `disco.events` table and creates and sends the appropriate discovery event to the probe for Tivoli Netcool/OMNIbus, nco_p_ncpmonitor process, which then forwards the event to the ObjectServer. You can control whether discovery events are generated by changing the value of the `m_CreateStchrEvents` field in the `disco.config` table. See also, AgentStatus, FinderStatus, and CreateStchTimeEvent stitchers.
DiscoShutdown	Activated when DISCO is shut down. Calls the RefreshDiscoveryTables stitcher.
ExampleContainment1	An example stitcher that could be modified to configure the containment model.
ExampleContainment2	An example stitcher that could be modified to configure the containment model.
FddiLayer	Deduces the FDDI layer topology.
Feedback	Sends device details back to the Ping finder to seed the discovery again.
FinalPhase	Activated in the final phase to implement the final stitchers.
FindAddressSpace	Identifies the address space of an IP address.
FinderStatus	This stitcher sends events to the `disco.events` table about the status of the finders. For each finder, the stitcher sends an event to indicate changes in the state of the finder; for example, if the finder has started, has finished, or has failed. See also, AgentStatus, CreateStchTimeEvent, and DiscoEventProcessing stitchers.
FindGatewayInterfaces	Identifies the gateway interface on NAT translation devices.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
FindPhysIpForVirtIp	Used in resolution of HSRP issues. Finds the physical IP address corresponding to a virtual HSRP address.
FnderProcToDetailsDesp	Processes entries in the `finders.processing` table, and sends the details to one of the following agents: • Details agent, if the device was discovered directly in the network. • CollectorDetails agent, if the record is a device discovered using an EMS collector.
FnderRediscoveryToCollectorFinder	Sends IP addresses or address ranges from the `finders.rediscovery` table to the Collectors. If the Collector server address or one of the devices that it collects matches the address or address range then the Collector processes it again.
FnderRediscoveryToPingFinder	Sends data from the `finders.rediscovery` table to the Ping finder.
FnderRetProcessing	Processes entities in the `finders.returns` table. Checks whether the device is in scope and moves this entry to the `finders.processing` or `finders.pending` table, depending on whether the discovery is in blackout state.
FullDiscovery	Determines whether a full discovery is to be run.
GetEntityNameByBase	For a given base name and interface index (or interface ID), this stitcher resolves the associated entity name.
GetEntityNameByIp	For a given address and optional address space, this stitcher resolves the associated entity name. An optional base name can also be specified to restrict the search.
GetBaseNameByIp	Returns the base name associated with the supplied IP address, or "" if none is found. If there are multiple matches then the first is used.
HandleIPMRouteDownstream	Processes the IPMRoute downstream routing data for the current device. It creates downstream route entities which populate the ipMRouteDownstream NCIM table. It also tracks end points required by the route, which are created later, and which MDT to associate the route with.
HandleIPMRouteUpstream	Processes the IPMRoute upstream routing data for the current device. It creates upstream route entities which populate the ipMRouteUpstream NCIM table. It also tracks end points required by the route, which are created later, and which MDT uses to associate the route with.
HubFdbToConnections	A precompiled stitcher that processes all of the connections for the Ethernet hubs. It also requires the connectivity information from the Ethernet switch discovery.
IlmiLayer	Creates the ILMI (Interim Local Management Interface) topology connections based upon ATM ILMI information.
InitiateNATGatewayDiscovery	Seeds the Ping finder with the NAT gateway addresses.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
InstantiationFilter	Determines whether a given entity is to be instantiated (that is, sent to MODEL), based on the instantiateFilter defined in the scope database. By default, the discovery filters do not filter out the Network Manager server. This is because this server typically also serves as the polling station for root cause analysis. In order for root cause analysis to work correctly, the polling station, and hence the Network Manager server, must be part of the topology. If you need to filter out the Network Manager server using the instantiateFilter modify the InstantiationFilter stitcher and remove the sections of code indicated by comments that prevent the Network Manager server from being filtered.
IPLayer	Creates the IP layer topology connections.
IpToBaseName	Populates the translations.ipToBaseName table with information from the AssocAddress agent.
IsForcedRediscovery	This stitcher is used to determine if a finder insert is part of a forced rediscovery. Forced rediscovery contrasts with reactive rediscovery, the mode that the Discovery Engine, ncp_disco, adopts after completion of a discovery. In this mode a device is typically only rediscovered if it is new or if the finder insert references a trap, thus suggesting that the entity has been modified. Forced rediscoveries are started using the Discovery Configuration GUI.
IsInMPLSScope	Determines if a given IP address is in the scope of devices considered to be valid CE devices connected to an inaccessible third-party MPLS PE device.
IsInScope	Used by other stitchers to check that an entity is within the scope of the discovery (that is, within the scope defined in the scope.zones table).
LLDPLayer	Determines connectivity information of remote neighbors based on data returned by the LLDP agent. **Note:** If connectivity is incorrectly displayed for a devices, then this might mean that the LLDP MIB on the network device is incorrectly populated. In some cases the relevant MIB data is incorrectly populated with device model number instead of a unique identifier. In this case the LLDP stitcher is unable to calculate LLDP connectivity correctly. To verify that this is the problem, for each of the devices that are not connected correctly you must examine the values of the LLDPChassisId field in the LLDP agent's LLDP.returns table. If you determine that the LLDPChassisId field values are not unique, then edit the LLDPLayer stitcher and set the processing method to a value of 2, by changing the following line in the stitcher: `int processingMethod = 2;`
MergeLayers	Merges the layer topologies.
ModifyIPContainment	Modifies the containment of IP interfaces on non-IP forwarding devices so that they are not upwardly connected. This modification is required to trace root cause.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
MPLSAddPathnames	Updates the MPLS interface records with information about path membership.
MPLSAddVPNNames	Determines what paths belong to which VPNs, and updates the MPLS interface records with the information about the VPN/Path membership.
MPLSCE	Tries to resolve CE to PE connectivity for VRF interfaces on a PE where the connecting CE has not been identified. It uses layer 3 information to try to find the correct connectivity.
MPLSPathDiscovery	Identifies labels switched paths (LSPs) between provider edge (PE) routers across an MPLS core. Starts path traces from the PE devices. The entry point stitcher sets up the path trace database, and begins a path trace for each possible path, calling other stitchers to update the records with path and VPN information.
MPLSProcessing	Determines which mode of MPLS discovery to perform based on the value of the field m_RTBasedVPNs in the disco.config table. • If m_RTBasedVPNs equals 1, then route target-based MPLS post-layer processing is performed. The MPLSProcessing stitcher calls the RTBasedVPNDiscovery stitcher to perform this processing. The MPLS discovery results in the ability to display an edge view only. • If m_RTBasedVPNs equals 0, then label switched path (LSP)-based post-layer processing is performed. The MPLSProcessing stitcher calls the MPLSPathDiscovery stitcher to perform this processing. The MPLS discovery results in the ability to display an edge view and a core view. This stitcher also performs the background processing required to generate service-affected events.
MPLStackProcessing	Ensures that any interfaces that are situated below a VPN supporting interface in the interface stack are marked as being part of the VPNs which flow through the higher interfaces.
NameResolution	Finds entities where the name has not been resolved and attempts to resolve the entity name based on the resolved names of the other interfaces of the device.
NamingFromLoopbackDetails	Provided there is a LoopBack agent running, this stitcher updates the names in the translations.ipToBaseName table. The management IP address of the device used by the poll policies is set to one of the loopback addresses, if Network Manager has confirmed that it has SNMP access.
NamingViaManagementInterface	Looks for management IP addresses from the translations.ipToBaseName and ensures the base address and name of an entity is that of the management server.
NATAddressSpaceContainers	Optional stitcher that builds NAT container objects holding devices within a particular address space and creates inserts into the workingEntities.finalEntity table for these NAT container objects. Also builds relevant entries into the workingEntities.containment table.
NATAgentRetProcessing	Processes the output from the NAT gateway agents.
NATFnderRetProcessing	Performs processing of NAT devices.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
NATGatewayRetProcessing	Used in discoveries involving NAT gateways where one or more of the management interfaces of the NAT gateway device is in private address space. This stitcher performs the processing necessary to determine whether each management interface is in public or private address space. This stitcher is called by the NATGatewayAgent agent and works with the `NATGateways.txt` file in `NCHOME/precision/etc`.
NATIpCheck	Resolves an issue where a NAT gateway adds all of its translated IP addresses to its own IP table.
NATTimer	Triggers rediscovery of NAT gateways.
NortelPassportLayer	Resolves the NortelPassport connectivity discovered by the NortelPassport agent.
OSPFLayer	Creates a topology of the OSPF routing within the network. This OSPF routing information is used by the DetermineOSPFDomains stitcher in order to tag devices and interfaces with OSPF domain information.
ParseASAMIfString	Parses the ASAM Interface description data into its component parts. Called from the ASAMIfStringLayer stitcher.
ParseZyxelIfString	Parses the ZYXEL Interface description data into its component parts. Called from the ZyxelIfStringLayer stitcher.
PeerBasedPWDiscovery	Used in discovery of enhanced Layer 2 VPNs on an MPLS core network. This stitcher identifies MPLS pseudowire connections retrieved by the Cisco MPLS agents and adds information about these connections to the relevant network entities for viewing in Topoviz. The information is stored as a pseudowire VPN and provides information about the two provider edge (PE) router ends of the pseudowire.
PIMLayer	Creates PIM Topology table based on remote neighbor data from PIM supporting agents. The topology data is used to populate the m_PIMAdjacency data, which in turn is used to populate the PIM Topology in NCIM
PingFinderScopeRefresh	Tells the Ping Finder to refresh its scope. This stitcher is activated by the Discovery Configuration GUI when you refresh the scope, ensuring that the Ping finder has an up-to-date scope.
PnniLayer	Creates the PNNI topology connections provided the connections at both ends have been discovered.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
PostLayerProcessing	A holder for all the functionality that is required following the creation of the layers. Calls the following stitchers: • AddGlobalVlans • AddSwitchRoutingLinks • AddUnconnectedContainment • BuildMPLSContainers • BuildVRAndVRFContainers • BuildVPNContainers • CreateTrunkConnections • CreateVlanEntity • MPLSAddVPNNames • MPLSPathDiscovery • MPLSInterfaceStackTrace • MPLSFindConnectionInStack • MPLSFindInterfaceInStack • MPLSAddPathNames • PVCPathMemberships • PVCTracePath • PVCProcessingRecord • PVCTraceAway • PVCTraceCrossConnected • PVCNamePath • PVCProcessedRecord • ProcessSwitchModules

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
PostScratchProcessing	A holder for functionality to occur following the creation of the scratch topology. Calls the following stitchers: • CreateNetworkManagementCards • InstantiationFilter: This stitcher is run as many times as necessary in order to check whether a given entity must be sent to MODEL. • SendTopologyToModel • AddTechnologyType This stitcher also calls the following stitchers that configure BGP-related information within the topology: • CreateBGPServices • CreateBGPProtocolEndPoints • CreateBGPTopology • CreateBGPAutonomousSystems • CreateBGPNetworksCollection This stitcher also calls the following stitchers that configure OSPF-related information within the topology: • CreateOSPFServices • SetOSPFServiceDesignatedStatus • CreateOSPFProtocolEndPoints • CreateOSPFNetworkLSAPseudoNodes • CreateOSPFPointToPointAdjacencies • CreateOSPFAreas • CreateOSPFRoutingDomains
PreProcessIGMPEndPointData	Creates and populates a temporary table consisting of end-point information for each IGMP-enabled interface and known groups. It also tracks the Multicast groups for which there is IGMP data. This data is used by other IGMP stitchers to create end point and group entities.
PresetLayer	Can be used to "preset" undiscoverable connections, if required. This stitcher is not used by default. This stitcher contains advanced configuration settings. Any changes must be made by certified personnel only.
ProcessQinQData	Processes QinQ data associated with interfaces and builds appropriate containment.
ProcessSwitchModules	Identifies which switch modules have their own IP addresses.
ProcRemoteConns	Takes a record containing a remote neighbor and processes remote connections if the agent that discovered it supports indirect connections.
ProfilingEndFinal ProfilingPhase1 ProfilingPhase2 ProfilingPhase3 ProfilingStartFinal	These stitchers populate the `disco.profilingData` table, providing data on discovery duration, memory usage, and a broad overview of the results of the discovery. This information is used in the estimation of scaling, and provides you with an overview of discovery performance.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
PruneSwitchConnections	This stitcher can be used as a way of improving switch connectivity in cases where the switches do not provide full connectivity information. This stitcher is not enabled by default, and must be enabled only on advice from IBM Support.
PVCNamePath	Adds the name of a PVC path to the internal `atmPVCs.memberships` database table.
PVCPathMemberships	Run automatically by `CreateScratchTopology.stch` during the discovery process. Uses the connectivity information from the Scratch Topology and the PVC information retrieved by the CiscoPVC agent to trace the PVCs across the network.
PVCProcessedRecord	Updates the `atmPVCs` database to indicate which record is currently being processed.
PVCProcessingRecord	Updates the `atmPVCs` database to indicate which record is currently being processed.
PVCTraceAway	Performs PVC tracing.
PVCTraceCrossConnected	Performs PVC tracing.
PVCTracePath	Performs PVC tracing for a given interface using the other PVC tracing stitchers to trace all the paths through the entire ATM section of the network.
PVCTraceTowards	Performs PVC tracing.
RebuildFinalEntityTable	This stitcher is very similar to the BuildFinalEntityTable. It also uses the entries in the `translations.ipToBaseName` table to populate the `workingEntities.finalEntity` table. The difference is that this stitcher is used in rediscovery mode rather than full discovery mode.
RecreateAndSendTopology	This stitcher is very similar to the CreateAndSendTopology.stch. It also activates the stitchers that create the topology and sends the final Scratch Topology to MODEL. The difference is that this stitcher is used in rediscovery mode rather than full discovery mode.
RecreateScratchTopology	This stitcher is similar to CreateScratchTopology.stch. The difference is that this stitcher is used in rediscovery mode rather than full discovery mode.
ReDoIpToBaseName	Refreshes the `translations.ipToBaseName` table.
RefreshDiscoveryTables	Refreshes the discovery database tables.
RefreshLayerDatabase	Refreshes a given layer topology database.
RefreshMPLSTEScope	Refreshes the scope of the StandardMPLSTE agent.
RefreshMulticastScope	Refreshes the scope of the StandardPIM agent.
RefreshSubnets	Refreshes a given subnet database.
RemoveDeviceFromTopology	Removes a device from the topology. The first argument of this stitcher must be the base name of the device to be removed.
RemoveInferredCEDuplicates	When the existence of a CE router is inferred, this stitcher removes potential duplicate devices from the topology.
RemoveOutOfBandConnectivity	Removes connectivity for out of band devices from the fullTopology.entityByNeighbor table.
RemoveOutOfBandRouterLinks	Removes router link connectivity for out of band devices from the scratchTopology.entityByName table.
RemoveWrongConnectionsToTA838	Removes wrong connections from Cisco 7609 and Cisco 3400 to Adtran TA838 devices.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
ResetNATMainNodes	Resets the IP of devices whose addresses have been translated by NAT from the private IP we use to resolve connectivity back to the public IP for monitoring. This allows the devices to be connected and visualized correctly and also remain accessible for monitoring purposes.
ResolveHSRPIssues	Checks for entities that have been discovered through their virtual Hot Standby Routing Protocol (HSRP) address. In that situation, the stitcher updates the discovery agent `returns` tables and the `translations.ipToBaseName` to show the correct physical interface.
ResolveVRRPAssocAddresses	Resolves issues caused by VRRP addresses. In such a situation, the stitcher updates the discovery agent `returns` tables and the `translations.ipToBaseName` to show the correct physical interface.
RestartDiscoProcess	Calls the **`restart_disco_process.pl`** script which stops the currently running discovery process and starts a new instance of it. It takes a single argument and a new full discovery is initiated by the newly started discovery process if the value is set to 1. If set to 0, then a full discovery is not initiated.
Restitcher	Re-stitches the topology together.
RTBasedVPNDiscovery	Discovers MPLS VPNs based on route target usage. This results in an edge view only which shows the MPLS core network with provider edge (PE) routers for VPNs and VRFs within the scope of the discovery. This view does not show the provider (P) routers within the MPLS core network and associated LSPs (label switched paths) that link these P routers. For each PE router discovered, Network Manager holds information on the route targets imported into and exported from that PE router. This enables you to identify which VPNs use which PE routers.
RTBasedVPNResolution	Uses the VRF data pre-processed by the RTBasedVPDiscovery stitcher to resolve VPNs based on Route Target import and export.
ScopeRefresh	Informs the finders and agents that require scope information when the scope table has changed.
SendToCollectors	Sends the supplied seed to the Collector finder for rediscover.
SendTopologyToModel	Sends the stitched topology to MODEL.
SerialLinkLayer	Determines connections from the data returned by the SerialLink agent.
SetOSPFServiceDesignatedStatus	Specifies whether or not the router running an OSPF service is a designated router or a backup router.
SONMPLayer	Determines connections from the data returned by the SONMP agent.
SubnetConnections	Creates subnet entities and inserts into each of the interfaces belonging to the subnet. At layer 3 level the interfaces within a subnet are all considered to be connected, so any connections not already discovered are added to the IP layer database.
SubnetToIPLayer	Adds default layer-three containment and/or connectivity.
SRPLayer	Builds the SRP layer to hold the containment information discovered by the SRP agent. In common with other layer stitchers, this stitcher receives input from relevant agents. This input consists of entity records containing local and remote neighbor data fields. The stitcher uses these records to work out the local and remote connections for each entity.

Table 149. List of Network Manager discovery stitchers (continued)

Stitcher	Function
SwitchFdbToConnections	Copies entries from the Switch agent `returns` tables to the `connections` table.
SwitchStpMltProcessing	Adds connections for all links in a multi-link trunk to an entityByNeighbor table.
SwitchStpToConnections	Builds a new layer based on the SwitchStp connectivity. Processes the data from the STP agent to create correctly named local and remote entity connection records in the stpTopology database. In common with other layer stitchers, this stitcher receives input from relevant agents. This input consists of entity records containing local and remote neighbor data fields. The stitcher uses these records to work out the local and remote connections for each entity.
SysNameNaming	Causes the system to name devices using the SNMP sysName where the data is valid. This is an optional stitcher that is off by default.
TagManagedEntities	Adds a tag to each of the interfaces of a main node indicating whether that interface should be monitored or not. This tag is in the `m_ExtraInfo` field and is called `m_IsManaged`. The tag can take the following values: • `0` - the interface is managed. This is the default. • `1` - the interface is unmanaged. This can be altered using the GUI. • `2` - the interface is unmanaged by the **`ncp_disco`** process. This cannot be altered using the GUI. • `3` - the interface is outside the discovery scope and is not to be polled. The `m_IsManaged` values for all interfaces in a main node are concatenated and stored in the `m_ExtraInfo` field for the main node, within `m_UnmanagedInterfaces`, using the format: `[<ifIndex1>, <IfIndex2>, ..... <IfIndexN>]`, where the ifIndices are the ifIndices of the interfaces you do not want the system to monitor. By default, the stitcher sets `m_IsManaged` to `0` for certain predefined types of interface, such as virtual interfaces. You can specify other interface types that you do not want the system to monitor by adding to a `where` clause within the stitcher.
TagManagementInterfaces	Tags the interface that has the IP address used as the main access IP address for a given entity. This stitcher is used in root cause analysis.
TraceRouteConnectivity	Updates the `IPLayer.entityByNeighbor` table with connectivity information retrieved from the TraceRoute agent returns data.
VRFBasedVPNResolution	Uses the VRF data pre-processed by the RTBasedVPDiscovery stitcher to resolve VPNs based on VRF names.
ZyxelIfStringLayer	Uses the ZYXEL ifDescr format to deduce connectivity.

Related concepts:

"Filters" on page 5
Use prediscovery filters to increase the efficiency of discovery and post-discovery filters to prevent instantiation of devices.

"Process flow for creating discovery events" on page 164
Discovery events are created during the discovery process showing the progress of agents, stitchers, and finders. These events are sent to and stored in Tivoli Netcool/OMNIbus and can be viewed using the Web GUI.

Related tasks:

"Setting discovery filters" on page 28
Use filters to filter out devices either before discovery or after discovery. You can filter out devices based on a variety of criteria, including location, technology, and manufacturer. Filters provide additional restrictions to those defined in the scope zones.

"Scoping discovery" on page 17
To scope the discovery, define the zones of the network (that is, subnet ranges) that you want to include in the discovery, and the zones that you want to exclude.

Appendix F. Types of traps

Traps are administrative messages sent from network devices such as routers that indicate that the device or its connections have started or stopped.

The Trap finder discovers devices by listening for SNMP traps and extracting IP addresses from those traps. The different types of traps are described in Table 150.

Table 150. Types of traps

Number	Name	Description
0	coldStart trap	A coldStart trap signifies that the sending protocol entity is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered.
1	warmStart trap	A warmStart trap signifies that the sending protocol entity is reinitializing itself such that neither the agent configuration nor the protocol entity implementation is altered.
2	linkDown trap	A linkDown trap is generated by the failure of a recognized communication link.
3	linkUp trap	A linkUp trap is generated when a communication link that was formerly down comes alive.
4	authenticationFailure trap	An authenticationFailure trap is generated by a protocol message that has not been authenticated by the recipient, for example, an incorrect password.
5	egpNeighborloss trap	An egpNeighborLoss trap signifies that an Exterior Gateway Protocol (EGP) neighbor for which the sending protocol entity was an EGP peer has been marked down and the peer relationship is no longer valid.
6	enterprise-specific trap	An enterprise-specific trap signifies that the sending protocol entity recognizes that an enterprise-specific event has occurred.

Appendix G. Network Manager glossary

Use this information to understand terminology relevant to the Network Manager product.

The following list provides explanations for Network Manager terminology.

AOC files
Files used by the Active Object Class manager, `ncp_class` to classify network devices following a discovery. Device classification is defined in AOC files by using a set of filters on the object ID and other device MIB parameters.

active object class (AOC)
An element in the predefined hierarchical topology of network devices used by the Active Object Class manager, `ncp_class`, to classify discovered devices following a discovery.

agent See, discovery agent.

class hierarchy
Predefined hierarchical topology of network devices used by the Active Object Class manager, `ncp_class`, to classify discovered devices following a discovery.

configuration files
Each Network Manager process has one or more configuration files used to control process behaviour by setting values in the process databases. Configuration files can also be made domain-specific.

discovery agent
Piece of code that runs during a discovery and retrieves detailed information from discovered devices.

Discovery Configuration GUI
GUI used to configure discovery parameters.

Discovery engine (ncp_disco)
Network Manager process that performs network discovery.

discovery phase
A network discovery is divided into four phases: Interrogating devices, Resolving addresses, Downloading connections, and Correlating connectivity.

discovery seed
One or more devices from which the discovery starts.

discovery scope
The boundaries of a discovery, expressed as one or more subnets and netmasks.

Discovery Status GUI
GUI used to launch and monitor a running discovery.

discovery stitcher
Piece of code used during the discovery process. There are various discovery stitchers, and they can be grouped into two types: data collection stitchers, which transfer data between databases during the data collection

phases of a discovery, and data processing stitchers, which build the network topology during the data processing phase.

domain
See, network domain.

entity
A topology database concept. All devices and device components discovered by Network Manager are entities. Also device collections such as VPNs and VLANs, as well as pieces of topology that form a complex connection, are entities.

event enrichment
The process of adding topology information to the event.

Event Gateway (ncp_g_event)
Network Manager process that performs event enrichment.

Event Gateway stitcher
Stitchers that perform topology lookup as part of the event enrichment process.

failover
In your Network Manager environment, a failover architecture can be used to configure your system for high availability, minimizing the impact of computer or network failure.

Failover plug-in
Receives Network Manager health check events from the Event Gateway and passes these events to the Virtual Domain process, which decides whether or not to initiate failover based on the event.

Fault Finding View
Composite GUI view consisting of an **Active Event List (AEL)** portlet above and a Network Hop View portlet below. Use the Fault Finding View to monitor network events.

full discovery
A discovery run with a large scope, intended to discover all of the network devices that you want to manage. Full discoveries are usually just called discoveries, unless they are being contrasted with partial discoveries. See also, partial discovery.

message broker
Component that manages communication between Network Manager processes. The message broker used byNetwork Manager is called Really Small Message Broker. To ensure correct operation of Network Manager, Really Small Message Broker must be running at all times.

NCIM database
Relational database that stores topology data, as well as administrative data such as data associated with poll policies and definitions, and performance data from devices.

ncp_disco
See, Discovery engine.

ncp_g_event
See, Event Gateway.

ncp_model
See, Topology manager.

ncp_poller
See, Polling engine.

network domain
A collection of network entities to be discovered and managed. A single Network Manager installation can manage multiple network domains.

Network Health View
Composite GUI view consisting of a Network Views portlet above and an **Active Event List (AEL)** portlet below. Use the Network Health View to display events on network devices.

Network Hop View
Network visualization GUI. Use the Network Hop View to search the network for a specific device and display a specified network device. You can also use the Network Hop View as a starting point for network troubleshooting. Formerly known as the Hop View.

Network Polling GUI
Administrator GUI. Enables definition of poll policies and poll definitions.

Network Views
Network visualization GUI that shows hierarchically organized views of a discovered network. Use the Network Views to view the results of a discovery and to troubleshoot network problems.

OQL databases
Network Manager processes store configuration, management and operational information in OQL databases.

OQL language
Version of the Structured Query Language (SQL) that has been designed for use in Network Manager. Network Manager processes create and interact with their databases using OQL.

partial discovery
A subsequent rediscovery of a section of the previously discovered network. The section of the network is usually defined using a discovery scope consisting of either an address range, a single device, or a group of devices. A partial discovery relies on the results of the last full discovery, and can only be run if the Discovery engine, `ncp_disco`, has not been stopped since the last full discovery. See also, full discovery.

Path Views
Network visualization GUI that displays devices and links that make up a network path between two selected devices. Create new path views or change existing path views to help network operators visualize network paths.

performance data
Performance data can be gathered using performance reports. These reports allow you to view any historical performance data that has been collected by the monitoring system for diagnostic purposes.

Polling engine (ncp_poller)
Network Manager process that polls target devices and interfaces. The Polling engine also collects performance data from polled devices.

poll definition
Defines how to poll a network device or interface and further filter the target devices or interfaces.

poll policy
Defines which devices to poll. Also defines other attributes of a poll such as poll frequency.

Probe for Tivoli Netcool/OMNIbus (nco_p_ncpmonitor)
Acquires and processes the events that are generated by Network Manager polls and processes, and forwards these events to the ObjectServer.

RCA plug-in
Based on data in the event and based on the discovered topology, attempts to identify events that are caused by or cause other events using rules coded in RCA stitchers.

RCA stitcher
Stitchers that process a trigger event as it passes through the RCA plug-in.

root-cause analysis (RCA)
The process of determining the root cause of one or more device alerts.

SNMP MIB Browser
GUI that retrieves MIB variable information from network devices to support diagnosis of network problems.

SNMP MIB Grapher
GUI that displays a real-time graph of MIB variables for a device and usse the graph for fault analysis and resolution of network problems.

stitcher
Code used in the following processes: discovery, event enrichment, and root-cause analysis. See also, discovery stitcher, Event Gateway stitcher, and RCA stitcher.

Structure Browser
GUI that enables you to investigate the health of device components in order to isolate faults within a network device.

Topology Manager (ncp_model)
Stores the topology data following a discovery and sends the topology data to the NCIM topology database where it can be queried using SQL.

WebTools
Specialized data retrieval tools that retrieve data from network devices and can be launched from the network visualization GUIs, Network Views and Network Hop View, or by specifying a URL in a web browser.

Notices

This information applies to the PDF documentation set for IBM Tivoli Network Manager IP Edition 3.9.

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
958/NH04
IBM Centre, St Leonards
601 Pacific Hwy
St Leonards, NSW, 2069
Australia
IBM Corporation
896471/H128B
76 Upper Ground
London
SE1 9PZ
United Kingdom
IBM Corporation
JBF1/SOM1 294
Route 100
Somers, NY, 10589-0100
United States of America

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The terms in Table 151 are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Table 151. IBM trademarks

AIX	iSeries	RDN
ClearQuest	Lotus	SecureWay
Cognos	Netcool	solidDB
Current	NetView	System z
DB2	Notes	Tivoli
developerWorks	OMEGAMON	WebSphere
Enterprise Storage Server	PowerVM	z/OS
IBM	PR/SM	z/VM
Informix	pSeries	zSeries

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Index

Printed in the Republic of Ireland

SC27-2762-04