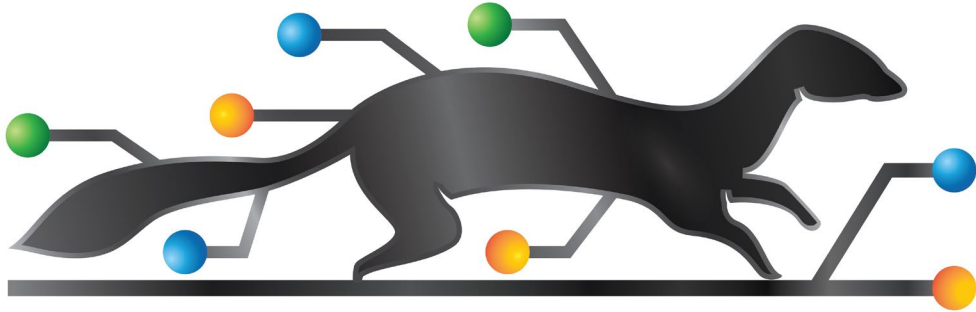


Network Ferret TM

User Guide

V12.3



Contents

Preface	i
Network Ferret License Agreement.....	i
License	i
About This Guide.....	i
Who is This Guide For?	i
Getting Additional Help.....	i
Getting More Information	i
Getting Support.....	ii
1: About Network Ferret	1
What is Network Ferret?.....	1
What is NMS Core?.....	1
Key Components.....	1
The Basic Engine.....	1
Advanced Discovery Extensions.....	1
Vendor Device Support.....	2
Configuration Files	2
Logs.....	2
Reports	2
2: Installing Network Ferret.....	3
Installation	3
Removing Network Ferret	3
3: Familiarizing Yourself with Network Ferret	4
File Locations	4
Running your first discovery	5
Editing Sample.cfg	5
Running Discovery.....	5
Typical problems.....	5
Monitoring Discovery	6
Viewing the Discovery Output	6
Changing a parameter.....	7
Experimenting with advanced features.....	8
4: Network Ferret CSV Reports.....	9
Basic Discovery Reports	9
raw.csv	9
hosts.csv.....	10
programs.csv	11
routerPorts.csv.....	11
Statistics.....	12
stats_raw.csv	12
Connectivity Reports	12
layer2_raw.csv.....	12
layer3_raw.csv.....	12
connectivity_raw.csv	12



intGrp_raw.csv.....	12
atm_raw.csv.....	12
5: Discovery Tools.....	13
Config File Manager.....	13
Setting the Root Directory.....	13
Window Panes.....	13
Filtering.....	14
Running Discovery.....	15
Other Apps.....	15
Saving Your Changes.....	16
Exporting.....	16
Log Tools.....	17
Log Formatter.....	17
Log Analyzer.....	17
Log Chopper.....	18
6: Wire Model Tools.....	19
Common File Menu Options.....	19
Connection Analysis.....	20
Node Analysis.....	22
VLAN Group Analysis.....	24
Browsing CSV Data.....	25
7: Device Analysis Tools.....	26
MIB Dump Manager.....	26
Entity MIB Dump Analyzer.....	26
MIB Definition File Scanner.....	26
VMWare Walk.....	27
8: Discovery Strategies.....	28
Discovering a Known Network.....	28
Advantages.....	28
Disadvantages.....	28
Seed Device Discovery.....	28
Advantages.....	29
Disadvantages.....	29
Discovering Via Connectivity vs Subnets.....	29
Advantages.....	30
Disadvantages.....	30
Discovering The Sphere of Influence of a Device.....	30
Discovering an Unknown Remote Office.....	30
Advantages.....	31
Disadvantages.....	31
Determining Device Uniqueness.....	31
Avoiding ICMP Pings.....	31
Advantages.....	31
Disadvantages.....	31
Avoiding Unreported Nodes.....	31
Controlling Traffic On The Network.....	32



Discovering Hosts On Non-Standard SNMP Ports.....	32
9: Configuring Network Ferret	33
Configuration Files	33
Configuration File Basics.....	33
10: Network Ferret Log Files	35
Message Format	35
Time.....	35
Message Type	35
Component.....	36
Address	36
Message	36
Progress Messages (P).....	36
Error Messages (E)	36
Debug Messages (. or w)	37
Formatting the Log	37
11: Running Network Ferret in a Firewalled Environment	39
Why Firewalls Are Different.....	39
Firewall Situations	39
Network Ferret Proxy	40
Cascading Proxies	41
Proxy Performance.....	41
12: Recording a Discovery	42
Recording a Discovery	42
Replaying a Discovery.....	42
Replaying a Discovery in Real Time	42
Replaying a Discovery in Natural Time	43
13: Topology.....	44
Discovery versus Topology	44
Definition versus Instance	44
How Topology is Stored	45
Directory Hierarchy	45
Working With Discoveries	46
Creating A Discovery Definition	47
Creating A Discovery Instance.....	47
Deleting Topology Information.....	48
Listing Topology Information.....	48
Comparing Two Discoveries	49
Appendix A – MIB Dumps.....	52
Appendix B – NetConf Dumps	53
Appendix C – Messages	54



Preface

This preface contains background information that you should know before using this Guide.

Network Ferret License Agreement

License

READ THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT SIGNED BY YOUR COMPANY BEFORE INSTALLING AND/OR USING NETWORK FERRET AND ANY ACCOMPANYING USER DOCUMENTATION. BY INSTALLING AND/OR USING THE SOFTWARE, YOU ("CUSTOMER") ARE ACCEPTING AND AGREEING TO THE TERMS OF THE LICENSE AGREEMENT.

About This Guide

The chapters of this guide contain explanations, reference information, and examples of how to use the Network Ferret Software.

Who is This Guide For?

This Guide is intended for all users of Network Ferret.

This Guide assumes that readers already understand:

- The basics of SNMP, Network Management and the need for an autodiscovery capability.
- The basics of using operating systems such as Microsoft Windows, Sun Solaris and RedHat Linux.

This Guide assumes readers already have:

- Access to a copy of the Network Ferret installation media.

Getting Additional Help

The following sources of additional assistance and information are available to all Network Ferret users.

Getting More Information

You can find additional information about Network Ferret in the following related publications:



- *Network Ferret Programmer Guide*: Intended for programmers who want to extend the functionality of Network Ferret using custom Java code.
- *Network Ferret Domain Model*: Intended for all users who want to understand the data that Network Ferret generates. This includes general models as well as models generated by vendor-specific device handling.

Getting Support

When contacting support, be prepared with the necessary information that will make handling your problem easier. You should have:

- `discoveryLog.txt` – this may be called something else by your application. This file can get rather large. If shipping the file to support, it is best to zip it up. If the problem is regarding a specific device, support may have you sort the log and pull out the information for that device rather than sending the entire log.
- `discoveryErrors.txt` – this may be called something else by your application.
- `discoveryExceptions.txt` – this may be called something else by your application.
- In some cases support may ask you to zip up the config, log and report directories.



1: About Network Ferret

This chapter introduces the Network Ferret autodiscovery engine and the basic concepts one will need to interact with the engine.

What is Network Ferret?

Network Ferret is an embeddable IP autodiscovery engine. Network Ferret is built to be a part of a larger management application such as an NMS, CMDB or an asset management application. Network Ferret was built to discover large networks quickly, efficiently and accurately. Not only does Network Ferret discover basic node information but it also discovers connectivity between Layer2 devices (switches and repeaters), Layer3 devices (routers), and other complex network relationships.

What is NMS Core?

With version 12 of Network Ferret, core functionality which is independent of autodiscovery has been separated out. This allows developers to leverage years of development experience for their own applications.

Examples of exposed functionality include: Ping, Credential management, MIB Dumps, SNMP Table walking, REST and NetConf access. The Programmers Guide provides detail on using NMS Core.

Key Components

The Basic Engine

The Basic Engine is responsible for individual node discovery and placing nodes in the proper subnets. The basic engine can be customized through configuration files or through custom Java programming called Basic Discovery Extensions.

Advanced Discovery Extensions

Advanced Discovery Extensions are responsible for determining connections between Nodes both physical (point-point) and logical. An Advanced Discovery Extension is fed from Basic Discovery and typically waits for basic discovery to complete before beginning its discovery.



Vendor Device Support

Information is mostly collected from devices using SNMP. Most of this information is defined in industry standard MIB files. However, there are some cases where standards are not defined and many cases where vendors choose not to follow the standards. Network Ferret has the capability to define custom logic for these cases.

Configuration Files

Network Ferret understands that technical customers and Systems Engineers in the field want to be able to configure systems using simple text files. Since Network Ferret has no GUI, all configurations are done through configuration files rather than through a GUI or programmatic means. Network Ferret has dozens of parameters that control the discovery process. You decide how much of this configuration to expose to your customers.

Logs

No two networks behave the same. Many factors can cause anomalous results during a discovery. Some of these factors include: SNMP timeouts, wrong community strings, firewalls, dropped pings and poorly implemented MIBs. Network Ferret maintains a comprehensive log which was designed such that you or even your customers can make some sense of what happened during a discovery.

Reports

Output in Network Ferret consists of reports created in CSV format suitable for viewing in a spreadsheet or importing into a database.

These reports are for evaluation purposes only. They would typically not be part of any product that a Network Ferret customer would create. Although customers seem to leave them on as they have become a useful debugging tool.



2: Installing Network Ferret

This chapter describes the installation procedure for Network Ferret.

Installation

See the readme.txt file that comes with the installation.

Removing Network Ferret

There are no files installed outside of the directory in which you did your Network Ferret install. So removing Network Ferret is simply a matter of deleting the directory into which you installed Network Ferret.



3: Familiarizing Yourself with Network Ferret

This chapter introduces the basic operation of Network Ferret and provides detailed information about the output of Network Ferret.

The install directory will be referred to as NFROOT. An operating system command line prompt will be referred to as a command prompt.

File Locations

All files are located under the installation directory (NFROOT).

\bin – contains scripts for running Network Ferret in its own Java VM. On Linux, this directory also contains the program AtiPrivilegedSocket which is critical for the functioning of ICMP (Ping) in Network Ferret.

\config – contains various configuration files. These are human editable text files that contain all of the parameters available to configure Network Ferret. More detail on the configuration files can be found elsewhere. This directory will also contain any Network Ferret Recorder (.rcd) files you create.

\config\vendor – contains the vendor specification (vsp) files which define the peculiarities of vendor devices. You should not have to worry about these files during the familiarization process.

\database – contains saved discoveries created by the Topology feature.

\doc – contains this document in PDF form, the Programmer and Domain Model Guides, the readme.txt and all of the Javadoc.

\java – contains the Network Ferret jar files, 3rd-party jar files, the source for examples and the source for constants (Factory names, Log messages).

\log – contains the Network Ferret log files. The log files are text based.



\report – contains the Network Ferret report files. These reports are CSV based (comma separated value) files. Any MIB dumps created will appear in this directory. WireModel.dat will also be here.

Running your first discovery

Network Ferret has many advanced discovery features such as discovering layer 2 and layer3 connectivity. These advanced features are disabled in this first run of discovery so the only output you will see will be basic node discovery. Advanced features will be enabled later in the familiarization process.

Editing Sample.cfg

Network Ferret is told what and how to discover through text-based configuration files located in NFROOT/config. You do not need to modify these files at this time.

Sample.cfg contains parameters that define the SNMP credentials to use and the subnet to discover. You should edit this file now. Use any text editor you like.

The other option is to use the ConfigManager application to create configurations and run discovery. See the [Tools](#) chapter.

For this first run, you should only pick one or two subnets to discover. As you get more familiar with Network Ferret you can expand the scope of a discovery.

Running Discovery

Open a command prompt and go to the NFROOT/bin directory. Run the discovery command with sample.cfg as the argument. The default config file defined in amtroot is examples.cfg which runs the example programs.

Typical problems

A common problem that occurs with a Network Ferret install is that Network Ferret is unable to open the ICMP socket for pinging. This is generally a problem on Linux machines because of a root or administrator privilege setting. See the readme.txt in the doc directory of the installation for details.



Monitoring Discovery

The script turns off all of the standard messages that normally come out of a script. The first thing you will see is the names of the configuration files being parsed.

Once Network Ferret has parsed the configuration files, you will see the Copyright notices. Then Network Ferret will open the log file: NFROOT/log/discoveryLog.txt and start writing to it.

Network Ferret produces three kinds of messages in the log. Progress (P) messages provide information about the progress of discovery. There should not be many of these messages. Error (E) messages report errors with the discovery. Finally, debug (. or w) messages provide a great deal of detail about what the engine is doing. These will be the most prevalent messages.

Warning messages (w) are for the embedding team, not for end users.

Progress messages will be written to the command prompt as well as the log. So you do not need to monitor the log file in order to know what the engine is doing. Error messages go to errors.txt as well as the log. Debug messages only go to the log.

Below is some sample progress output from a small discovery. The important messages to note are the last three. The difference between basic discovery and discovery extensions will be explained later. Since there are no extensions running during this first discovery, the three messages will come out simultaneously.

```
Config: d:\amt\amt_nms\config\defaults.cfg
...
Config: d:\amt\amt_nms\config\test.cfg
135 - Network Ferret NMSCore v12.2 (April, 2024) - Copyright (c) 2024 by Alternative
Management Technology Inc. All Rights Reserved
117 - Network Ferret v12.2 (April, 2024) - Copyright (c) 2001-24 by Alternative Management
Technology Inc. All Rights Reserved
130 - Starting on subnet 172.16.10.51 num: 9 from: Seeds hop: 0 with mask: subnets active: 0
108 - Done spec: 172.16.10.51 - number of hosts created: 6 via Ping 6 via ARP: 0
130 - Starting on subnet bucket num: 4 from: Seeds hop: 0 with mask: subnets active: 0
108 - Done spec: bucket - number of hosts created: 4 via Ping 4 via ARP: 0
104 - DONE ALL HOSTS IN BASIC DISCOVERY - minutes: 0 subnets: 0 hosts: 10
103 - DONE ALL EXTENSIONS
102 - DISCOVERY COMPLETE. minutes: 0 errors: 0
```

Viewing the Discovery Output

Once Network Ferret has completed its first discovery, go to the NFROOT/report/. You will see several files in this directory. Open the file raw.csv. Your spreadsheet application should open and



display the discovery data. This is the same data that is sent to the embedding application via API calls.

It is not that important to understand the details of the data model at this point in time. You will see many bits of information that look familiar.

At this time you can also browse NFROOT/log/discoveryLog.txt although much of it will not make any sense to you right now. The one thing to note is that after the timestamp you will see a one-character field. This field will contain either a P for Progress messages, an E for Error messages or a dot (.) for debug messages. The dot was used to make it easier to visually scan for progress and error messages. Tools are provided to format the log for easier viewing. These tools are described later.

Changing a parameter

Network Ferret is configured via several configuration files. See the [Configuring Network Ferret](#) chapter. In the default installation these files are found in the NFROOT/config directory.

Defaults.cfg will be read first, then any agent-specific config files and finally your config file. In this case, this is sample.cfg.

Edit sample.cfg and edit the line:

```
IPDDNSLookup = false
```

Either change false to true or put a # (comment indicator) at the front of the line. The effect is the same. Save the file.

Browse discovery.cfg and you will see the same parameter with the value set to TRUE. The value in sample.cfg will override the value in discovery.cfg.

Run Network Ferret again using the same subnet(s) you used in the first run. Browse raw.csv and you will notice that the Node objects now have dns and domain attributes (assuming your DNS is properly set up).

You could also have edited discovery.cfg but it is better to make changes for a specific discovery in your configuration file rather than in the defaults.



Experimenting with advanced features

Once you are comfortable with the basic discovery, you can enable one of the advanced discovery features. Edit the sample.cfg and change the parameter, IPDEExtension.ext-layer2.shouldRun, from false to true.

Run Network Ferret with this new configuration. In the previous discoveries you saw the following progress messages appear simultaneously.

```
09:29:01 P main          noAddr          - DONE ALL SUBNETS IN BASIC DISCOVERY - minutes: 0 subnets: 1
hosts: 4
09:29:01 P main          noAddr          - DONE ALL EXTENSIONS
09:29:01 P main          noAddr          - DISCOVERY COMPLETE. minutes: 0 errors: 0
```

With the Layer 2 extension running, there will now be other progress messages between the DONE ALL SUBNETS message and the DONE ALL EXTENSIONS message. This assumes that there were some layer 2 devices actually discovered.

Once the discovery is completed, review the layer2Raw.csv file in the report directory. This file will provide a list of layer 2 ports and their remote connections. The remote connections will either be simple MAC addresses of hosts or references to other layer 2 ports in the case of cascaded layer 2 devices.



4: Network Ferret CSV Reports

Network Ferret generates a number of csv (comma separated value) files. In a default installation, all reports can be found in the report directory under NFROOT. The location of reports can be changed in the configuration file.

To completely understand the reports you should read the Network Ferret Domain Model document. It describes all of the objects and relationships that will be found in these reports.

The data in these files is in comma-separated value format. You can certainly view these files in a basic text editor but a spreadsheet application would make it much easier to understand. All further explanation assumes the use of a spreadsheet application that has basic sorting and formatting capability.

The content of the `_raw.csv` files is exactly what will be passed back to an embedding application via the API (except that comma's in the data are replaced by blanks so the spreadsheet formats properly). These "raw" files are important in helping us debug connectivity issues.

There is also a file called `wireModel.dat` that is generated in the report directory. This is important for debugging. See `discovery.cfg` for how to enable/disable wire model generation.

Basic Discovery Reports

raw.csv

Open `raw.csv` in your spreadsheet application. The first thing to do is format the data. Highlight the first seven columns (up to column G). Do a single sort of the data using columns B (most significant), C and D (least significant). Resize the columns to taste.

The output of Network Ferret is a data model of the networks discovered. This data model contains two types of objects: entities and relationships. Entities are objects such as nodes, interfaces and ports. Relationships define a connection between two entities. The Network Ferret Domain Model Guide explains the model in



detail. Entities use four columns in the csv file and relationships use six.

Entity columns:

Shipment type – will be the same for all rows.

Host name – the IP address, DNS name of the node in which this entity was discovered. The name for nodes with multiple addresses could be the SNMP sysName of the node.

Class name – the class of the entity. For example, Node, Board and Interface.

Unique Identifier – the unique identifier within that node for the entity.
Unique identifiers include IP addresses and SNMP indexes.

Other attributes – this value contains a list of attribute=value pairs. Examples are: community=communityString and vendor=vendorName.

Relationship columns:

Shipment type – will be the same for all rows.

Host name – the IP address or DNS name of the node in which this entity was discovered. The name for nodes with multiple addresses could be the SNMP sysname of the node.

Class name – the class of the relationship. For example, Z-contains or Z-access. The Z- is prepended in the csv file solely for sorting purposes. The normal output of Network Ferret does not contain the Z-.

Parent Class – the class name of the parent entity of the relationship.

Parent Unique Identifier – the unique identifier of the parent entity.

Child Class – the class name of the child entity of the relationship.

Child Unique Identifier – the unique identifier of the child entity.

Below is some sample raw.csv output for a basic non-SNMP node.
SNMP nodes will contain much richer information.

192.168.0.1	Interface	1	{snmpType=0 snmpIndex=1 overIndex=0 type=unknown}		
192.168.0.1	IP	192.168.0.1	{mask=255.255.255.0 address=192.168.0.1 subnet=192.168.0.0}		
192.168.0.1	Node	192.168.0.1	{discoveryIP=192.168.0.1}		
192.168.0.1	OS	1	{}		
192.168.0.1	Z-interface	Interface	1	IP	192.168.0.1

hosts.csv

Hosts.csv provides a simple listing of every IP Address found during discovery. The following columns are provided:

Subnet – the subnet discovered.



IP – the IP Address on the subnet.
Host – the host in which this IP Address lives.
Capabilities – provides some clue as to what this host does.

Note that hosts/subnets will appear in this file that will not appear in the subnets file. This is because addresses will be found inside of multi-address nodes that are on subnets that are not being discovered. Since the subnet is not being discovered, an accurate utilization cannot be reported in the subnets file.

programs.csv

Programs.csv provides a simple listing of every Program found during discovery. The following columns are provided:

Type – the type of the Program such as webserver or ftp or telnet.
Port – the IP port the Program was found on. Some programs are found via other means and will not have a port.
Host – the host in which this Program lives.
Capabilities – provides some clue as to what this host does.

routerPorts.csv

Programs.csv provides an inventory of all router interfaces found during discovery. The following columns are provided:

Router – the name of the Node.
Vendor – this is the vendor attribute from the Interface object. If there was no vsp file for a particular router the field will be empty.
Type – this is the type attribute from the interface object. This is vendor-specific information so if there was no vsp file for a particular router the field will be empty.
Interface Name – this is the description attribute from the Interface object.
Interface Type – this is the type attribute from the Interface object.
Interface Speed – this is the speed attribute from the Interface object.
Interface Index – this is the SNMP index from the Interface object.
MAC – this is the layer 2 MAC address from the Interface object. Only LAN interfaces will have a MAC address.
IP Addresses – a list of 0, 1 or more IP Addresses that sit on top of this interface.



Statistics

stats_raw.csv

As of 11.0, SNMP statistics are collected on each address being queried. The stats are separated by discovery type (Basic, L2, etc.). No totals are supplied.

The format of the file is obvious.

Connectivity Reports

See the Domain Model Guide for a definition of what each report generated.

layer2_raw.csv

Switch – switch and switch – host connectivity

layer3_raw.csv

Router to router connectivity – both physical and logical connections such as BGP, OSPF, etc.

connectivity_raw.csv

General protocol connectivity such as LLDP, CDP, etc.

intGrp_raw.csv

Connections determined by active polling of certain interfaces.

atm_raw.csv

Connections between ATM switches. These connections include physical switch to switch connections and not logical connections across many switches.



5: Discovery Tools

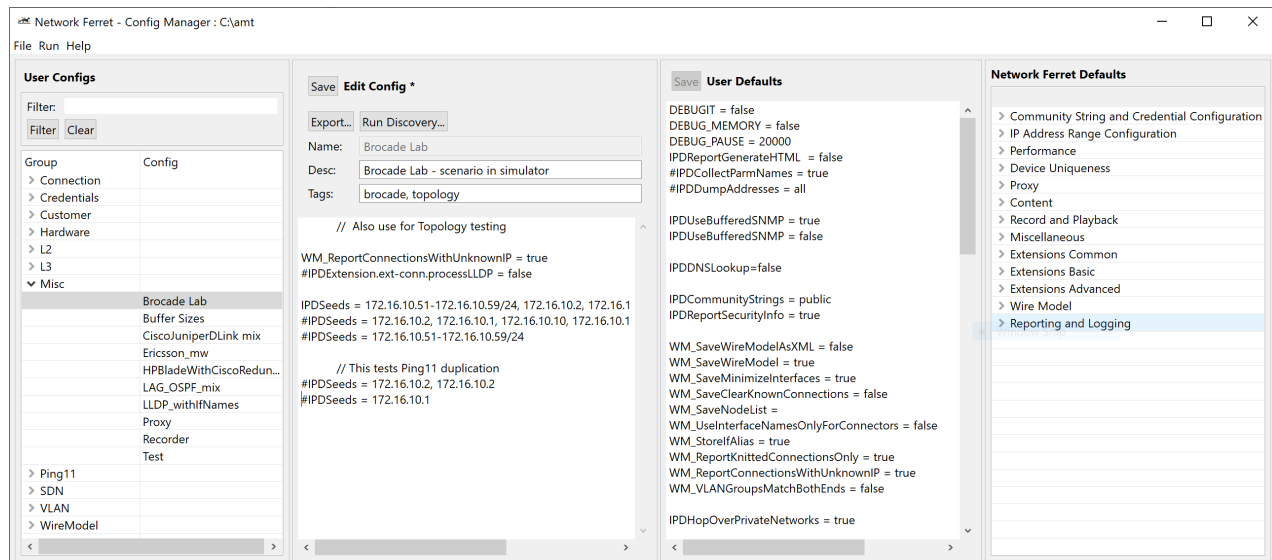
There are several tools that can be used to run and analyze discovery. Contact us if you want to run these. Most are not included in the standard distribution. The Log formatting tools are in the standard distribution.

Config File Manager

If you frequently run Network Ferret standalone to test various network scenarios, you probably have a big config file or many little ones.

The Config File Manager app provides a database to maintain configurations. The main class is:

`com.logikos.graphics.discovery.ConfigManager`



Setting the Root Directory

The first thing you must do is File->Set Root Dir. It is assumed that there are the usual Network Ferret subdirectories and that the config contains defaults.cfg. Then we look for the file configs.dat. If it does not exist, it is created.

Window Panes

Network Ferret Defaults



This pane shows the data from defaults.cfg. This data cannot be edited in this application. The categories shown come from the <Group> tag in defaults.cfg. These are here for reference. You can right click on a line and copy the text to the clipboard or send it to the Filter field in the User Configs pane.

User Defaults

This pane contains data from the config database. These are config parameters that are included in every config that you run/export. These override the Network Ferret defaults and they can be overridden by the same parameter in a specific config.

User Configs

This contains data from the config database. There is a two-level hierarchy. Group and Config. Every config must belong to a group. Right click (or double click) on a config to load it into the neighboring Edit pane.

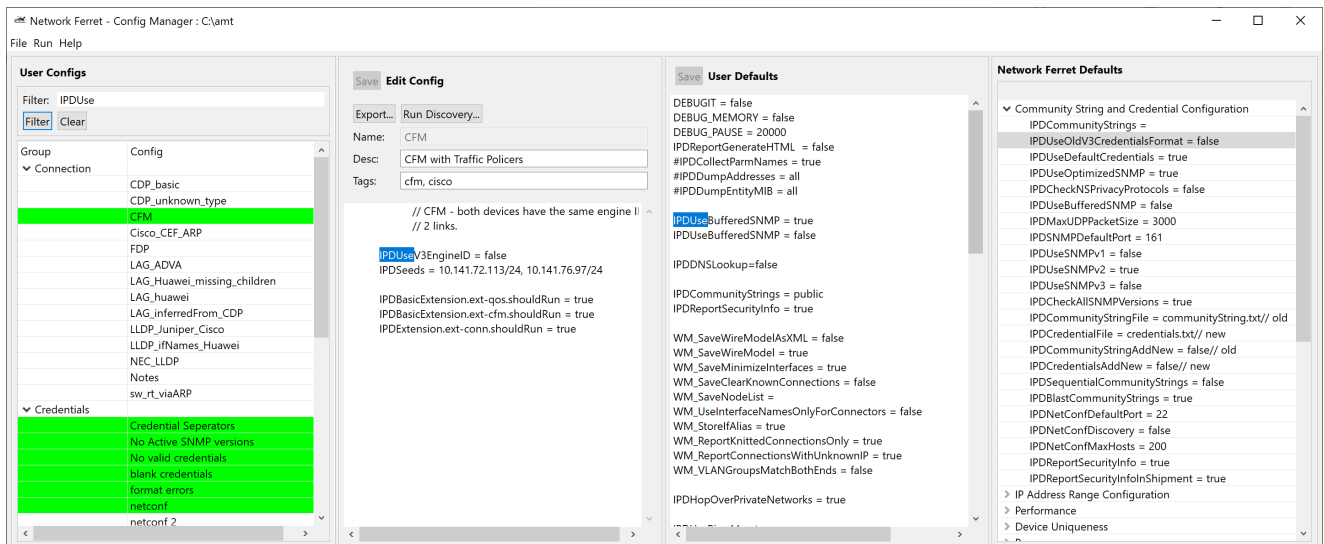
Edit Config

Edit a config. A config can have a description and a set of tags.

Filtering

The User Configs tree in the left pane has a Filter field and button at the top. Enter any text, select the Filter button and each pane will highlight the text if it is found.

The User Config tree will highlight any config containing that text. The Edit Config and User Defaults text windows will highlight the first instance of the filter text. The Network Ferret defaults will highlight the first line in the tree containing the text.



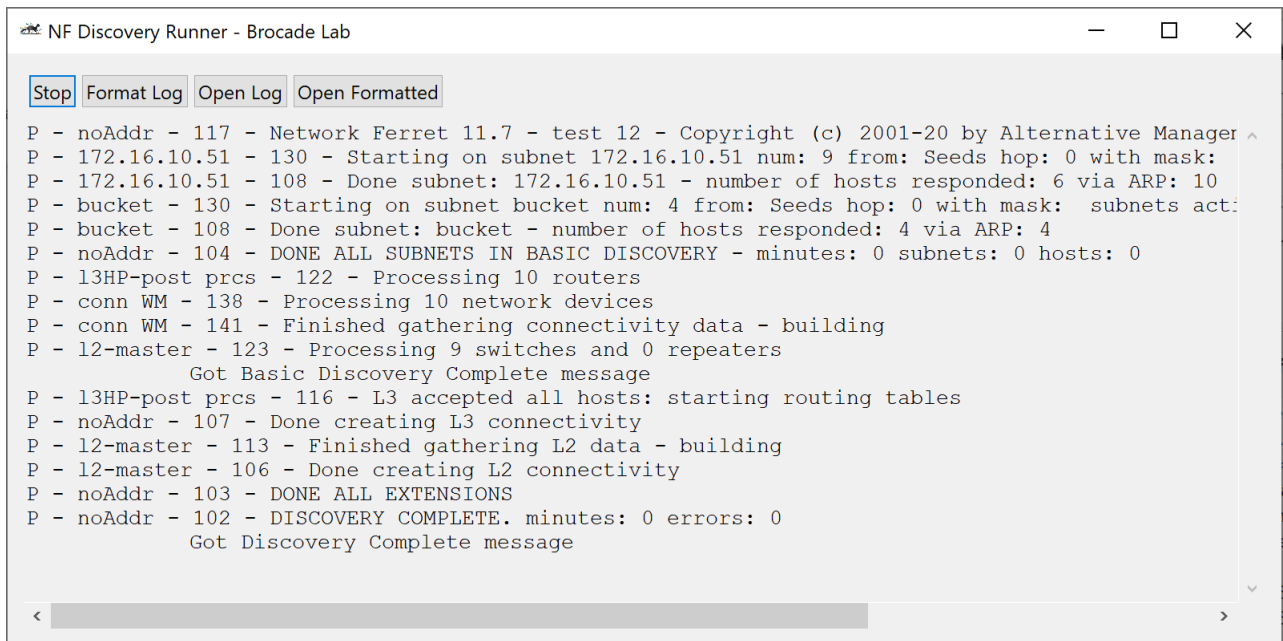
Running Discovery

Discovery can be launched from the Run menu or with the Run Discovery... button in the Edit Config pane.

For config parameters, the discovery will receive, as a single Properties object, the User Defaults plus the current config being edited. Network Ferret defaults are not used. The Network Ferret discovery jar contains all of the defaults from defaults.cfg.

It is very important to note that the discovery receives the contents of the panes regardless if you saved them or not. This allows you to quickly test several scenarios without having to always save your changes.

A new window is created.



The screenshot shows a window titled "NF Discovery Runner - Brocade Lab". At the top, there are four buttons: "Stop", "Format Log", "Open Log", and "Open Formatted". Below the buttons is a text area containing a log of discovery steps. The log starts with "P - noAddr - 117 - Network Ferret 11.7 - test 12 - Copyright (c) 2001-20 by Alternative Manager" and continues with various steps like "Starting on subnet 172.16.10.51", "Done subnet: 172.16.10.51", "Starting on subnet bucket", "Done subnet: bucket", "DONE ALL SUBNETS IN BASIC DISCOVERY", "Processing 10 routers", "Processing 10 network devices", "Finished gathering connectivity data - building", "Processing 9 switches and 0 repeaters", "Got Basic Discovery Complete message", "L3 accepted all hosts: starting routing tables", "Done creating L3 connectivity", "Finished gathering L2 data - building", "Done creating L2 connectivity", "DONE ALL EXTENSIONS", and "DISCOVERY COMPLETE. minutes: 0 errors: 0". The log ends with "Got Discovery Complete message".

Stop – stops the discovery.

Format Log – formats discovery.txt into discovery.formatted.txt the same as the format_entire_log script in the bin directory.

Open Log – open the un-formatted log.

Open Formatted – opens the formatted log.

Other Apps

The three main Wire Model apps are available to run from the Run menu. The Entity MIB Dump analyzer is also available.



Saving Your Changes

We do our best to make sure you save changes. There are save buttons in the User Defaults and Edit Config panes.

Any changes to the User Configs tree will be saved automatically and cause a reload of the Edit Config and User Defaults panes. So be careful that you have saved any edits you wanted to save before making changes to the tree.

Exporting

There are three reasons to export the config database.

One, in case a programming error is made and the data file cannot be reloaded. We still need to create an option to import!!

Two, to share a config with someone else.

A single file will be created that contains User Defaults at the top of the file and then a section for each config. This file is suitable to use as regular config file input to Network Ferret. Note that each config has the tags IGNORE and IGNORE_END around it. To “activate” a given config, simply comment out (#) the IGNORE line.

Three, to use a config when running discovery from within a development environment such as Eclipse. In this case, export a specific config (File menu, button in Edit pane or right-click in tree view). This will export the User Defaults and the selected config to a file suitable to be used by discovery.



Log Tools

Log Formatter

```
com.logikos.discovery.message.AtlIPDLogFormatter
```

```
logDirectory
```

```
logFile
```

```
outputFile
```

```
[host:address|progress|progress:address|error]
```

There is a script called `format_entire_log` in the `bin` directory which runs this program.

If only three parameters are provided, the output file will contain the entire contents of the `logFile` sorted by the third column of the `logFile` which is usually an IP Address.

If `host:address` is provided, then only lines with that IP address **anywhere in the line** will be included in the output file.

If `progress:address` is provided, then only Progress lines with that IP address **anywhere in the line** will be included in the output file.

If `progress` or `error` is specified, then only progress or error lines will be in the output file. `Errors.txt` already includes all of the error lines so there is no point in using that option here.

You cannot run this on a giant log. It will blow up with out of memory. If it blows up, use the Log Chopper below.

Log Analyzer

```
com.logikos.discovery.message.AtlIPDLogAnalyzer
```

```
logDirectory
```

```
logFile
```

```
outputFile
```

```
[hang|time|nc]
```

You can copy/modify the `format_entire_log` script from above.

If there are only three parameters, then some analysis will be done which we have been coding on the fly so we will not tell you what it is. Please use a fourth parameter.



If “hang” is specified, the logFile will be sorted by address and then each address analyzed to see if it did not finish discovery. The analysis will be written to outputFile.

If “time” is specified, the analysis is looking for time gaps > 20 seconds. A gap is a period of time where nothing is written to the log. The two log lines on either side of the gap will be written to the outputFile along with the gap size.

If “nc” is specified, NetConf connections will be analyzed.

Log Chopper

```
com.logikos.discovery.message.AtlIPDLogChopper
```

```
logDirectory  
logFile  
outputFile  
#linesPerFile (no comma as below)  
match1  
match2
```

Use LogChopper when the log file is far too large to load into a basic text editor.

200,000 lines per file generates a file of about 20meg. Loadable into Wordpad. Notepad can only handle 4-5 meg well.

X number of outputFiles will be generated (outputFile1-outputFileX). In addition, error messages and loopmon lines will be written to their own files.

If match1 [and match2] are specified, then any line containing that text (or both if match2 is specified) will be written to the file called outputFile_other.

If match1 is specified, the log will not be chopped. Only the _other file will be generated.

You cannot run LogFormatter (above) on a giant log. It runs out of memory. To look at a specific address, use the IP address for match1.



6: Wire Model Tools

The wire model file (wireModel.dat) is generated in the report directory along with raw.csv. We have created several GUIs to browse this data. We are not GUI masters so feedback on functionality is welcome but please no remarks on the aesthetics. We know it isn't pretty.

Common File Menu Options

The File menu contains some common options across the GUIs.

Load Wire Model

Load a Wire Model into the GUI. If the window was opened from another window, the Wire Model should already be loaded.

Prepare raw.csv

Right-clicking on IP addresses in the GUIs generally gives you a menu option to browse the raw.csv data for that address. However, you first need to choose this menu option which reads raw.csv and chops it up into a CSV file per IP address.

Knit Wire Models

This does the same function as the wmKnit script in the bin directory. First, select a root directory which must contain the config, log and report subdirectories. Next, select a config file. This is necessary for the logging/reporting functions to work. Finally, choose a name for the knitted wire model.

All wire models (.dat files) in the report subdirectory will be read in and knitted. Any .dat file that is already a wire model group will be ignored.

The knitted wire model group will be written out to the report subdirectory. The report directory will also contain the usual CSV files that a discovery would generate. The log directory will also contain the usual files.



Connection Analysis

See the wmConnAnalysis script in the bin directory.

File Help

Nodes: 10
Interfaces: 272
Routers: 10
Switches: 7

IPs	Type	Ints
172.16.10.1	r	10
172.16.10.10	rs	78
172.16.10.11	rs	53
172.16.10.2	rs	68
172.16.10.51	r	6
172.16.10.52	r	7
172.16.10.53	rs	10
172.16.10.54	rs	11
172.16.10.56	rs	12
172.16.10.59	rs	7

IP Address: 172.16.10.1 Interface index/name: 0 Analyze

Interface Lists: All Bottoms Connections

SysName: vyatta OID: 30803

C	MAC	BP	VLAN	Type	In...	IPs	PhyCon	LogCon	Root
				_loopback		[127.0.0.1]			1
*	0CC47ACE586E			ethernet					11
*	0CC47ACE586F			ethernet					12
*	0CC47ACE57EE			ethernet					13
*	0CC47ACE57EF			ethernet		[172.16.10.1 172....	172.16.10.2 - bp: 2 - if: 2 (SINGLE)		14
*	0CC47ACE57F0			ethernet		[2.16.254.128.0.0....	172.16.10.10/148 (LLDP)		15
*	0CC47ACE57F1			ethernet		[2.16.254.128.0.0....	172.16.10.11/60 (LLDP)		16
*	0CC47ACE57F2			ethernet		[2.16.254.128.0.0....	172.16.10.56/201711616 (LLDP)		17
*	0CC47ACE57F3			ethernet	c-	[10.10.10.1 10.10....			18
*	0CC47ACE57EE			ethernet	c-	[150.254.156.217 ...			19
*				_other		[172.16.50.1 172....			21

Columns

C – an asterisk (*) if the interface has a connector

MAC – the MAC address of the interface or blank

BP – the bridge port index or blank

VLAN – a VLAN ID or the word “multiple” or blank.

Type – the interface type from the ianaifypes file

Info – a code for which advanced discoveries worked with this interface.

IPs – the list of IPs directly associated with this interface

PhyCon – the list of physical connections directly associated with this interface.

LogCon – the list of logical connections directly associated with this interface.

Root/Child1/Child2/etc – An indication of where this interface sits in the hierarchy of the interface being analyzed. For most menu options, all of the interfaces will be in the root column.

First do File->Load Wire Model....

This will give you the list of IPs on the left. Right-click on a given IP and select one of the menu options to load data into the interface pane. The other option is to type the IP address into the entry field in the interface pane and select one of the buttons.

Buttons

Once a node is loaded into the interface list pane, the All/Bottoms/Connections/Analyze buttons at the top of the pane are available.



All – lists all interfaces in the node.

Bottoms – lists all interfaces with a connector (* in C column).

Connections – lists all interfaces that have a connection.

Analyze - an interface must be selected in the list below or you can type in a specific interface number. Analyzing an interface generates an interface stack of all interfaces above and below the interface being analyzed. This is where the Child1/Child2/etc. columns get utilized.

Interface Right-click Options

Analyze – same functionality as the Analyze button.

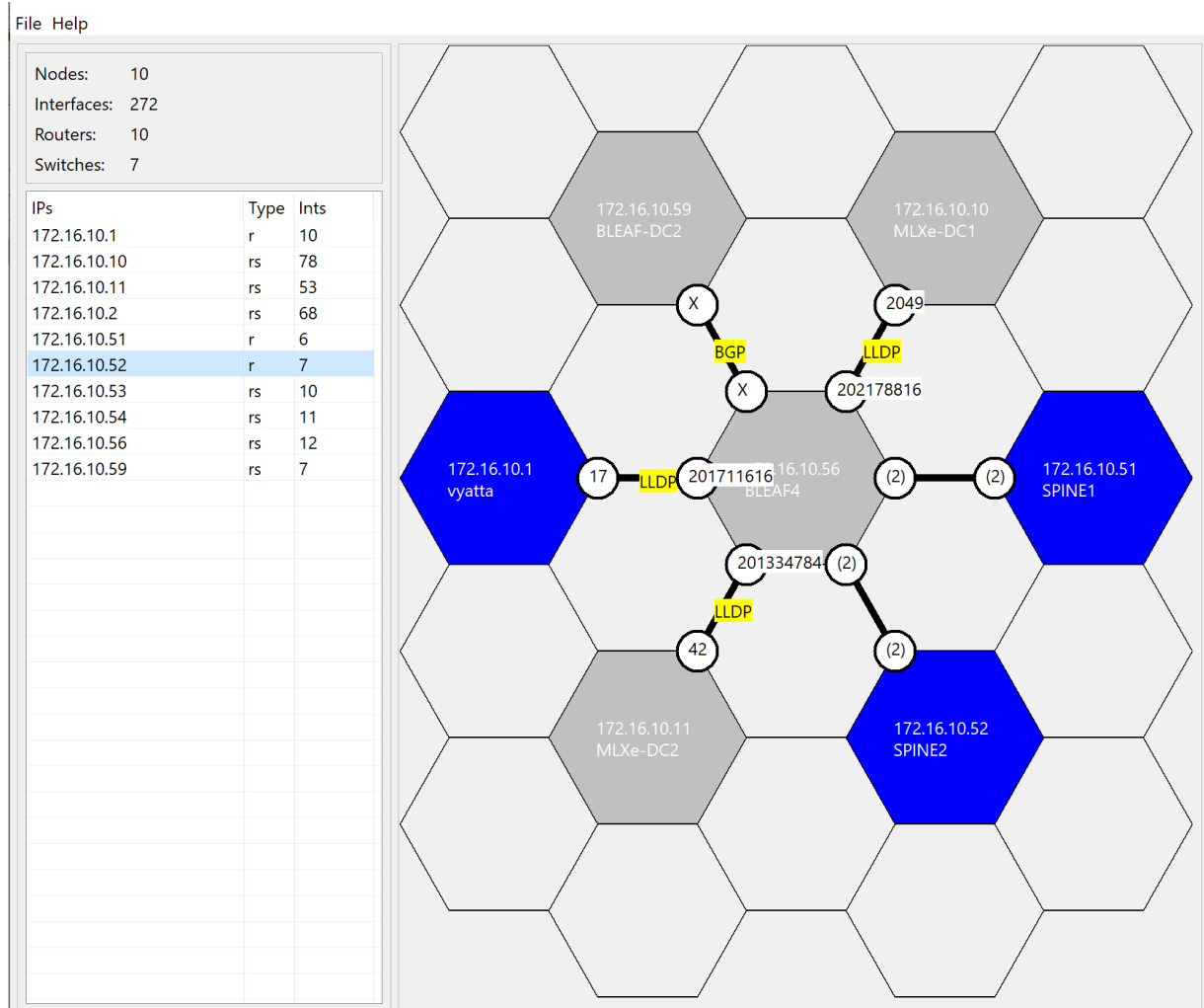
Load Connection in Other Pane – if the interface has a connection, the node/interface on the other end will be loaded into the other pane and Analyzed.

Show All Connections To Other End – if the interface has a connection, then the interface list will be filtered to show only interfaces that have a connection to the same far end node.



Node Analysis

This GUI is started from the File menu in the Connection Analysis GUI. If an IP is selected from the IP list in the Connection Analysis GUI, that IP will be the focus of the graphic. If no IP is selected, then this GUI will start with an IP list on the left and an empty graphic.



This GUI is showing connections between nodes. This is not a general network browsing GUI. It is for diagnosing problems with discovery. The first obvious problem with this GUI is that if a node has connections to more than 6 other nodes, this breaks down.

Right-click on any node to make it the center node.

The connection endpoints show the interface involved on the connection, the number of connections if there are multiple (#) or



an X for logical connections at the node level where no interface is involved (this will be a -1 in newer versions).

Where a link specifies a single connection between nodes, it is labelled with the protocol/method used to make the connection.



VLAN Group Analysis

This GUI is started from the File menu in the Connection Analysis GUI or from the script `wmVLANGroupAnalysis` in the bin directory.

Network Ferret - VLAN Group Analysis : C:\Users\ykostes\wireModelScenarios\brocadeLab\wireModel.dat

File Help

Groups:	Nodes:
6	10
Routers:	Switches:
10	7

VLAN	#SW	#RT	#SG
1	9	0	8
10	5	0	5
1002	6	0	6
4044	4	0	3
4093	6	0	6
4095	8	0	7

Node Type	Node	Int Type	Int	Other	Connections
Switches	172.16.10.59				
		Trunk			
			202178816		172.16.10.11/641 (LLDP)
	172.16.10.11				
		Trunk			
			1		
	172.16.10.10				
		Trunk			
			1		
	172.16.10.56				
		Trunk			
			202178816		172.16.10.10/2049 (LLDP)
Routers					

Group ID	Type	Info
4044.17	Switches	
		172.16.10.59
		172.16.10.11
	Routers	
4044.18	Switches	
		172.16.10.10
	Routers	
4044.19	Switches	
		172.16.10.56
	Routers	

The pane on the left shows each VLAN, the number of switches and routers associated with the VLAN and how many subgroups were created. Remember that two nodes can have the same VLAN ID but the VLANs are actually different VLANs.

So if there are multiple subgroups, either there are multiple VLANs using the same ID or Network Ferret did not find all of the necessary connections to realize they are the same VLAN.

The middle pane shows all of the switches and routers associated with the VLAN. The right pane shows each subgroup.



Browsing CSV Data

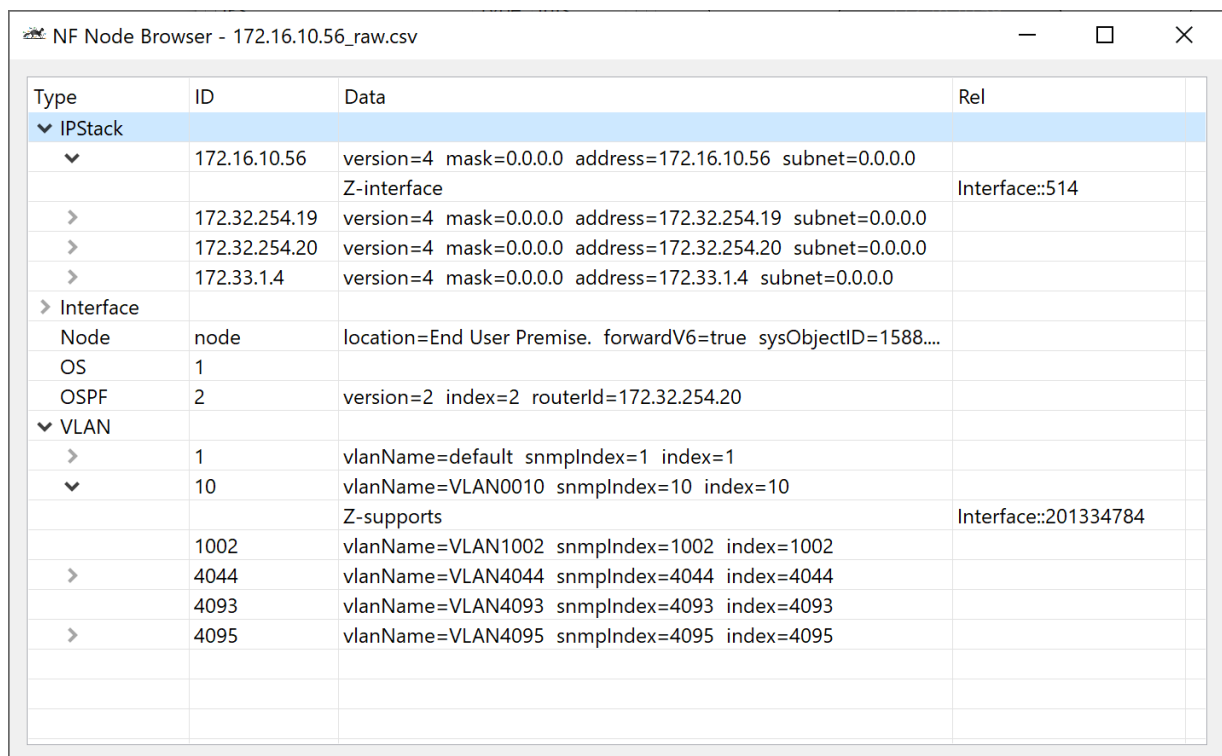
From the File menu of any of the GUIs, choose Prepare raw.csv. This will load raw.csv and generate a CSV file for each node in the file. The raw.csv file must be in the same directory as wireModel.dat.

You can actually run AtilPDCSVChopper independently. It is in the same Java package as the log analysis tools above. It takes three parameters:

The directory where the CSV can be found
The CSV file name
An optional exclude list [hw,misc,mpls,vlan,ip,l3]

Exclude object types that are not part of the issue you are looking at. Sometimes these objects can be rather voluminous and get in the way of browsing.

Once this is done, you can right-click on any IP in the GUIs and choose Browse raw.csv data.



The screenshot shows a window titled "NF Node Browser - 172.16.10.56_raw.csv". It contains a table with four columns: Type, ID, Data, and Rel. The table is organized into expandable sections like IPStack, Interface, and VLAN.

Type	ID	Data	Rel
▼ IPStack			
▼	172.16.10.56	version=4 mask=0.0.0.0 address=172.16.10.56 subnet=0.0.0.0	
		Z-interface	Interface::514
>	172.32.254.19	version=4 mask=0.0.0.0 address=172.32.254.19 subnet=0.0.0.0	
>	172.32.254.20	version=4 mask=0.0.0.0 address=172.32.254.20 subnet=0.0.0.0	
>	172.33.1.4	version=4 mask=0.0.0.0 address=172.33.1.4 subnet=0.0.0.0	
> Interface			
Node	node	location=End User Premise. forwardV6=true sysObjectID=1588....	
OS	1		
OSPF	2	version=2 index=2 routerId=172.32.254.20	
▼ VLAN			
>	1	vlanName=default snmplIndex=1 index=1	
▼	10	vlanName=VLAN0010 snmplIndex=10 index=10	
		Z-supports	Interface::201334784
	1002	vlanName=VLAN1002 snmplIndex=1002 index=1002	
>	4044	vlanName=VLAN4044 snmplIndex=4044 index=4044	
	4093	vlanName=VLAN4093 snmplIndex=4093 index=4093	
>	4095	vlanName=VLAN4095 snmplIndex=4095 index=4095	

The GUI will allow you to browse down one level of relationship.



7: Device Analysis Tools

As part of developing an application with Network Ferret embedded, you may decide that you want to do your own analysis of devices or use a MIB Simulator to help with your own development/QA.

There are several applications that are part of a separate distribution. Contact us if you are interested.

MIB Dump Manager

MDM is used to catalog MIB dumps that you collect from your customer sites or that come with your SNMP simulator product.

Entity MIB Dump Analyzer

The Entity MIB is one table that requires much analysis. While it is simple in concept (objects, containers) that simplicity allows vendors to do many different things with it.

Network Ferret attempts to fit the various Entity MIB strategies into a single data model that is output to the embedding application. But sometimes it does not work so well and some analysis is required.

For example, Ericsson microwave devices have a shadow tree in the Entity MIB hierarchy that actually represents the Entity MIB in the remote microwave that it is connected to.

MIB Definition File Scanner

Network Ferret is constantly dealing with vendor-specific MIBs and we often have to read MIBs to try to find some piece of data that was left out of the standard MIB implementation for the vendor.

This MIB file scanner works together with MIB Dump Manager.



VMWare Walk

VMWare is a bit special. It has its own data model (see Domain Model Guide). All VMWare objects are discovered in an advanced discovery processor using VMWare's API which is in `vijava.jar`.

The VMWare data model accessible via their API is very complex. Clients who want a particular piece of information reported by Network Ferret often have a hard time figuring out where to find it!

We have created an experimental VMWare Walk application (see the `readme.txt` file in the section:



8: Discovery Strategies

Networks are different and administrators know differing amounts about their network. Network Ferret provides multiple strategies for discovering a network. Originally, Network Ferret was based on the administrator knowing exactly what subnets they wanted to discover. Network Ferret has added features which allow the discovery of a somewhat unknown network.

Discovering a Known Network

Use this strategy when you know exactly what subnets you want to discover and not discover.

Use the IPDSeeds, IPDIncludeSubnets and IPDExcludeSubnets parameters to define exactly what to discover. You only need to specifically exclude a subnet if it lies within a larger range that you want to include. Make sure IPDMaxHops is set to 0.

It is also possible to specify a host file with a discrete list of host addresses and/or FQDNs.

Advantages

- No unwanted subnets will be discovered.

Disadvantages

- You must know your network.
- Will be very inefficient if anything larger than Class C subnets are used on WAN router links. See the strategies that employ expansion.

Seed Device Discovery

This strategy is similar to many other existing discovery engines. A single device is specified and all subnets/addresses within a given number of hops will be discovered.

Use the IPDSeeds parameter to configure the seed device address(s). IPDMaxHops needs to be set to something larger than 0 and one or both of IPDExpandUsingSubnets and IPDExpandUsingConnectivity needs to be enabled. Use IPDIncludeSubnets/IPDExcludeSubnets to limit what is discovered.



Advantages

- Easy to configure. One must know only the address of a single device that supports SNMP.
- This strategy can avoid using ICMP Pings.

Disadvantages

- Potential to discover unwanted subnets. Avoid this by using the IPDIncludeSubnets/IPDExcludeSubnets parameters.
- Potential to roam outside your network. Avoid this by using the IPDExcludeSubnets and IPDHopOverWANLinks parameter.
- Potential to not discover everything you wanted. Avoid this by setting the IPDMaxHops parameter to a very high number.
- Will be very inefficient if anything larger than Class C subnets are used on WAN router links.

Discovering Via Connectivity vs Subnets

In well-managed environments and/or environments that mostly contain routers and switches and few hosts, it may be more efficient to discover via connectivity rather than pinging entire subnets found in router IP address tables.

IPDExpandUsingConnectivity and IPDExpandUsingSubnets work in conjunction with the IPDMaxHops parameter.

IPDExpandUsingSubnets simply expands to the subnets found in the IP address tables of routers.

If IPDExpandUsingConnectivity is enabled, then a special processor is started which queries the following protocol tables looking for specific addresses to expand to:

OSPF, BGP, LDP, CDP, LLDP, GVRP and possible others.

Any address found in these tables will be following the same pinging rules as any other address. Network Ferret does not assume these addresses exist.

If IPDExpandToARPAAddresses is enabled (the default is TRUE), then addresses in router ARP caches will also be discovered.

For purposes of hop counts, any switch discovered via a switching protocol (GVRP, ST, LLDP) from another switch will use the same hop count. This is so you do not have to specify a high max hops number just to ensure discovery of a chain of switches.



Advantages

- Easy to configure. One must know only the address of a single device that supports SNMP.
- This strategy can avoid using ICMP Pings.
- This strategy avoids pinging possibly large, sparsely populated subnets.

Disadvantages

- Depending on which protocols are active (LLDP, CDP), there is a chance to miss switches since a switch's IP address will not show up in a router ARP cache unless someone is talking to the management address of that switch. If switches are on one or management subnets, then this can be worked around by specifying these subnets in the IncludeSubnets along with a seed router or any other initial subnets.
- Potential to roam outside your network. Avoid this by using the IPDExcludeSubnets and IPDHopOverWANLinks parameter.
- Potential to not discover everything you wanted. Avoid this by setting the IPDMaxHops parameter to a very high number.

Discovering The Sphere of Influence of a Device

Suppose a device changes and you only want to discover that device and everything interacting with it so you can update your database with any changes. Also suppose that the device is a router.

In this case you do not want to discover the IP addresses in the ARP cache because they most likely are not interacting with the router. In this case, set IPDExpandToARPAAddresses to false.

Discovering an Unknown Remote Office

Use this strategy when you want to discover one or more remote offices in your network but are not sure of the composition of those offices and do not want to discover any core parts of the network.

This strategy assumes that the remote office uses no internal WAN router links.

The seed router approach or a known subnet approach could be used. In both cases the IPDMaxHops value would be set high and the IPDHopOverWANLinks parameter would be set to false to contain the discovery within the remote office.



Advantages

- Contains a discovery to a geographic location.

Disadvantages

- Will not work if the location contains internal WAN links. In this case the IPDExcludeSubnets parameter must be used to prevent the discovery from moving outside the location.

Determining Device Uniqueness

Please see the discovery.cfg section on device uniqueness.

Avoiding ICMP Pings

Network Ferret prefers to ping all possible addresses on a subnet. However, some administrators do not like that. Use the IPDUseARPNotPing parameter set to true to avoid pinging addresses on LAN segments. Ping will still be used on WAN segments.

Advantages

- Avoids sending out massive amounts of pings on large, yet sparse, segments such as a class B with a few hundred active hosts.

Disadvantages

- The ARP cache is not guaranteed to contain all hosts on the segment. Especially other routers on the segment. Only use this option if your particular situation permits it.

Avoiding Unreported Nodes

There are certain situations where ICMP packets can be dropped or lost. This can happen in a very busy network or with specific devices that cannot handle a burst of ARP traffic.

In these situations increasing the timeout or decreasing the maximum pings will not help because the device never received the packet in the first place.

In the case of a device that cannot handle a burst of ARP traffic, you need to utilize the IPDPingDelay parameter to limit the number of pings per second on the network. Remember that a device must analyze any ping request just to find out that the request is not for itself.



In the case of a busy network where packets are being dropped, it is unfortunate that you actually have to increase the traffic to get a successful discovery. You don't want to increase the maximum pings outstanding (i.e. number of hosts being pinged concurrently) but you do want to increase the number of retries (IPDPingRetries) for a given host and drop the IPDPingTimeoutIncrement to 0 or something very small. Basically you are hammering a device with pings in the hopes that one will make it through.

Controlling Traffic On The Network

It is possible to control the amount of ICMP and SNMP traffic that Network Ferret generates. IPDPingDelay and IPDSNMPDelay are used to limit the packets per second placed on the network.

For example, the parameters are in milliseconds so a value of 10 for IPDPingDelay would place at most 100 pings per second on the network.

The various "Max" parameters such as IPDMaxPings should no longer be used and at some point in the future they will be removed.

Discovering Hosts On Non-Standard SNMP Ports

See the IPDHostFile config file parameter.



9: Configuring Network Ferret

Network Ferret uses simple text-based configuration files as input. This mechanism was chosen for two reasons. First, there are too many parameters to pass in as arguments. Second, experience has shown that field personnel/users love to tweak parameters by hand.

As the programmer integrating Network Ferret, you can expose as much or as little of the configuration files as you choose. You can also choose to use a Properties object rather than a file.

While there are many configuration parameters, your end user should not have to change many of them.

The main configuration file is called defaults.cfg and can be found in the default installation in NFROOT/config. Each run of Network Ferret also takes as a parameter another configuration file which contains two things. It must contain the specific addresses/subnets to discover in this run and the community strings/v3 credentials to use. It may also contain overrides to any parameters found in defaults.cfg. As an aside, since the lines in the configuration files are read sequentially, one could override parameters in defaults.cfg by simply restating them farther down in the defaults.cfg file. This is not a recommended strategy if humans are going to be editing this file.

Configuration Files

defaults.cfg – system-wide defaults – used by ALL agents.
pingAgent.cfg – defaults for the ping Agent
credentialAgent.cfg – defaults for the credential Agent
mibDumpAgent.cfg – defaults for the MIB Dump Agent
discovery.cfg – defaults for Network Ferret

There is also some discussion in the Programmer Guide.

Configuration File Basics

Configuration files are simple text files that use a key=value paradigm. XML was not used. While XML is great for



programmatic exchange of information, it tends to be too verbose for human consumption.

Below is a section from defaults.cfg.

```
# Should discovery look up DNS names or just use IP addresses
IPDDNSLookup=true
```

- All white space is stripped from the beginning and end of each line.
- All white space is stripped from around the equals sign (=).
- Blank lines are allowed and encouraged.
- A comment line starts with the # or with a forward slash (/).
- Comments at the end of a line with a key=value pair should begin with //.
- All keys and values are case sensitive.

Note that all of the defaults defined in defaults.cfg are reflected in the code so, technically, defaults.cfg is not necessary. You may want to use defaults.cfg as the defaults that you ship to your customers. The only exception to this are the definitions of the webserver, ftp and telnet ports in the Port discovery extension. These are not reflected in the code.

ALL CONFIGURATION PARAMETERS ARE NOW DOCUMENTED IN DEFAULTS.CFG.



10: Network Ferret Log Files

This chapter describes the content of the Network Ferret log files.

Network Ferret creates three log files for each discovery run. The name and location of the files are configurable via the configuration file. See the chapter on configuring Network Ferret. The default file names are `discovery.txt`, `errors.txt` and `exceptions.txt` and the default location is the log directory under `NFROOT`.

Message Format

Network Ferret generates a fixed-field log file. Some sample lines are below.

```
00:10:14 P main      noAddr      - 117 - Network Ferret 4.5.0...
00:10:14 . main      noAddr      - Java Vm Version: 1.4.2_04
14:38:28 P subnet    163.39.186.128 - 109 - Done with subnet-37    minutes: 1 hosts: 23
14:38:29 . node      163.39.248.209 - Start build - name: fred.whatever.com
14:38:29 . node      163.39.248.209 - capabilities: router,snmp
14:38:29 . int       163.39.248.209 - build interface type: ether address: 00024A650000
```

Each line contains the following fields:

Time

The Time field represents the time the message was generated not the time it was written out. For all intents and purposes, these two values are the same anyway because the log messages are not queued to a separate thread.

No date is written out at this time since our experience has shown no discovery to take longer than a day.

Message Type

This is a one-character field with the following values:

- P – Progress message
- E – Error message
- W – Warning messages – for programmers, not end users
- . (dot) – Debug message



Component

This field defines the component reporting the message. The component could represent a running thread or a domain object. For the most part these names are cryptic and will only make sense to Network Ferret developers or to those with a great deal of experience with Network Ferret.

Address

The IP address associated with the message. This address could be a host address, a subnet address or simply “noAddr” in cases where the message is about the system in general such as startup messages.

The advanced extensions will use an address for messages generated while collecting data. When they analyze the data and produce their output they will use a special tag for the address. Since the advanced extensions deal with connectivity information across nodes it is important that the messages be together rather than spread across each host.

This field has expanded to accommodate IPv6 addresses.

Message

A message will never span lines. This is because there are utilities that sort the messages and having messages that spanned lines would cause problems.

Progress and Error messages will begin with a number. This is to allow support teams to easily identify a message if they have been translated into another language.

Progress Messages (P)

Progress messages tell you how discovery is moving along. Progress messages are written to the log, stdout and available programmatically.

Progress messages tend to be subnet level messages as progress on individual hosts would be too voluminous. A full listing of progress messages can be found in the Appendix.

Error Messages (E)

Error messages do the obvious. They tell you when errors have occurred. There are two types of errors. There are errors



pertaining to the data being discovered such as a device not containing basic MIB variables or an interface index not being found for a given IP stack. There are also internal program errors such as null pointers and such.

For device errors the debug messages in the log generally give some indication of what is happening. Many times a MIB walk is required of the device in order to diagnose the problem and in many cases there is nothing that can be done. The device just behaves the way it behaves.

For program errors, in addition to a line in the regular log file, the Java stack is written out to exceptions.txt. There is a numeric identifier which links the two together. This was done so the regular log file could easily be sorted.

Error messages are written to both the regular log and to a special error log called errors.txt. Errors.txt just provides an easy way of looking at all errors without having to look through the entire log.

A full listing of known error messages can be found in the Appendix. All device errors will be known but program errors tend to be unknown. If they were known, then they would have been fixed.

Debug Messages (. or w)

Debug messages are by far the most voluminous messages in the log. They are very helpful when trying to figure out anomalous data reported about a device. For this reason it is wise not to disable the writing of debug messages even though the feature exists. Sometimes anomalous behavior is caused by timing issues or by network conditions at the time and recreating it could be difficult.

Warning messages (w) are messages that programmers need to be aware of but should not be shown to end users. They are errors but not an error that an end user would understand or do anything about.

Formatting the Log

Network Ferret runs many threads so messages about a number of devices are interspersed in the log. Sometimes you need to view the log in the order in which messages were written in order to analyze timing issues.



There are times when the log needs to be analyzed for the information regarding a specific device. The log as written is not conducive to looking at all of the messages for a given device.

A script called `format_entire_log` is provided in the `bin` directory. This script will read in the time-based log, sort it and write a new log sorted by IP address and then by time within each address. The default name of the formatted log is `discovery.formatted.txt`.

The script assumes a default installation so if you have made any changes you should edit the script. The script is merely calling a Java class so this functionality is also available programmatically. More detail can be found in the API sections.



11: Running Network Ferret in a Firewalled Environment

This chapter describes how to set up Network Ferret so firewalled environments can be discovered accurately.

The Proxy is NOT PART OF V12. No one ever used it so we removed it.

Why Firewalls Are Different

A firewall's purpose is security which means that many things that are taken for granted in a normal environment do not hold true in a firewalled environment.

ICMP and SNMP packets may not be allowed to pass through a firewall. Even if they are allowed, other problems exist. For example, DNS translation may not be available for hosts inside the firewall except from inside the firewall.

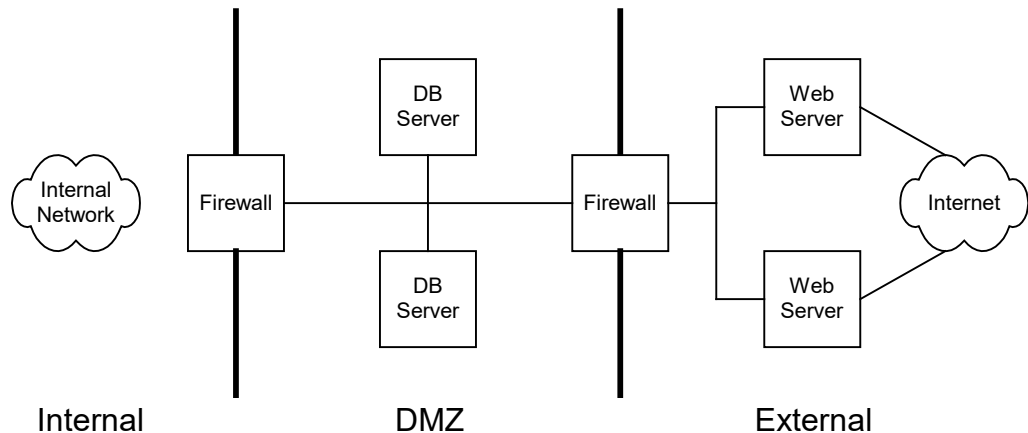
Network Address Translation is the process of hiding the true IP address of a host. An address of 192.168.20.1 inside the firewall may appear as 10.10.10.10 outside the firewall. The NATed addresses will not appear anywhere in the MIBs of the devices thus causing confusion with the discovery logic.

If there are multiple levels of firewall, communications typically cannot span multiple firewalls. Devices in one area can only communicate with devices in an adjacent area.

Firewall Situations

Assume the typical eCommerce situation below. The Internal network is not permitted to communicate with the External network and visa versa. All communications must go through the DMZ. The DMZ gear can only communicate with certain hosts on the Internal network. Both firewalls are doing NATing.





A centralized discovery engine, no matter where it is placed, could not discover the entire network. A discovery engine placed on the External network certainly could not do discovery, as ICMP and SNMP would never be allowed through the firewalls.

Only a cascading discovery engine can discover the entire network. A simple client-server architecture with the server placed in the Internal network still could not discover the External network because the Internal network cannot communicate directly with the External network.

Network Ferret Proxy

The Network Ferret Proxy is a subset of Network Ferret that is installed in the firewalled environment. The proxy does not contain any discovery logic. It consists of basic communications code that forwards ICMP, SNMP and DNS requests to devices in the firewalled environment. Since this code will rarely change, Network Ferret upgrades are very simple. Only the main Network Ferret installation needs to be upgraded.

The proxy is a long running process. It should be started once and left alone. The script `commproxy` starts the proxy process. The script takes one optional parameter (80 is the default) which is the port the proxy should listen on for requests.

The configuration file parameters, `IPDCommHost` and `IPDCommPort`, tell Network Ferret where the proxy is located. `IPDCommPort` must match the port number used to start the proxy.

For each ICMP or SNMP socket that Network Ferret opens (about 12), a TCP connection over the defined port will be created between Network Ferret and the proxy. The proxy will receive a



request from Network Ferret and forward it on to the desired address. The proxy will also listen for responses from the firewalled devices and forward them back to Network Ferret.

Cascading Proxies

For the example at the beginning of the chapter, a single proxy would not suffice. A proxy is required in the DMZ network as well as in the External network. And due to security policy, Network Ferret can only communicate with the External proxy through the DMZ proxy.

Cascading proxies is easy. Simply start the DMZ and External proxies and then place their addresses and ports in order in the configuration file.

Suppose that the DMZ proxy was started on port 80 on the host dmz1 and the External proxy was started on port 8080 on the host external1. The configuration file parameters would look like the following:

```
IPDCommType = remote
IPDCommHost = dmz1, external1
IPDCommPort = 80, 8080
```

If more than two levels of proxy are required, simply add them to the list.

Proxy Performance

Using a proxy will greatly slow the speed of discovery because all requests are funneling through a single machine. Tests have shown a 4-8 times increase in discovery times using the proxy. Cascaded proxies will be even slower. For this reason, the proxy should only be used to discover small firewalled environments.

Network Ferret does not batch requests to the proxy and the proxy does have any logic to retransmit after timeouts. There are no traffic volume gains to be had when using the proxy. Therefore, the proxy should not be used as a means to lower discovery traffic over a skinny WAN link.



12: Recording a Discovery

This chapter describes how to record a discovery and replay it. This is useful for developers and QA staff who need to regression test the embedding application or who need to recreate/test a bug fix.

Recording a discovery could also be useful in a support situation. For this reason, with Network Ferret 4.1 the record capability has been made part of the engine and not a separate standalone feature.

Recording a Discovery

Recording a discovery is simply a matter of setting some parameters in the configuration file.

```
IPDRecorderFile = a file name  
IPDRecorderType = record
```

The Recorder registers its interface with Network Ferret and receives all of the API calls. The Recorder writes every API call to your file with a timestamp. The file is created in the config directory.

Replaying a Discovery

Replaying a Discovery in Real Time

Replaying a discovery in real time means that the recorded file will be played back as quickly as it can be read. So if a real discovery took an hour, the playback may only take 10 seconds.

To replay a discovery in real time set the following parameters in the config file:

```
IPDRecorderFile = the record file name  
IPDRecorderType = playback  
IPDRecorderMode = realTime
```



The default mode is `realTime` so you do not have to specify it.
Make sure that your previously recorded discovery file is located in the Network Ferret config directory.

Replaying a Discovery in Natural Time

Replaying a discovery in natural time means that the recorded file will be played back exactly as it occurred. So if a real discovery took an hour, the playback will take an hour.

To replay a discovery in natural time set the following parameters in the config file:

```
IPDRecorderFile = the record file name
IPDRecorderType = playback
IPDRecorderMode = naturalTime
```

Make sure that your previously recorded discovery file is located in the Network Ferret config directory.



13: Topology

Discovery versus Topology

The difference between a Discovery and a Topology is that one is physical while the other is logical. A Discovery instance represents the data collected during a single run of Network Ferret. A Topology instance is a logical entity which is a collection of one or more Discoveries.

It is expected that end users will think at the Topology level while administrators will have to think at both the Discovery and Topology levels. Due to firewalls or corporate procedures or any number of other factors, it may require multiple discoveries to gather all of the information about a network (i.e. topology) to be managed.

At this point in time only the Discovery capability is available in Network Ferret. In other words, there is currently a one-one relationship between a Discovery and a Topology.

Definition versus Instance

Topology information is first organized by definitions and then within each definition there are multiple instances or versions. Suppose the network had to be discovered in three stages, the core, workgroup1 and workgroup2.

Each of these three stages would constitute a Discovery Definition. Within each Discovery Definition, Network Ferret is going to be run one or more times. Each run of Network Ferret would create a Discovery Instance of that particular Discovery Definition. You would then be able to compare run2 against run1 and see what the differences are.



Logically, this is what it would look like:

```
Discovery
  Core
    Version1
    Version2
    Version3
  Workgroup1
    Version1
    Version2
  Workgroup2
    Version1
```

How Topology is Stored

One of the design goals of Network Ferret is to keep it as lightweight as possible. Therefore topology information is stored in the file system rather than in a database. The tradeoff to this is that the topology feature is limited in the size network it can reasonably work with. Much depends on the speed of the computer, the speed of the disk and the file handling capabilities of the OS, but at this point in time it is suggested that the topology feature be used on networks of 1000 nodes or less.

The topology information is stored as one file per node plus extra files for the advanced discoveries (L2, L3, etc.). The config files and logs for a discovery are also stored.

Directory Hierarchy

Topology is stored under a main directory defined by the embedding application. Assume for now that this directory is:

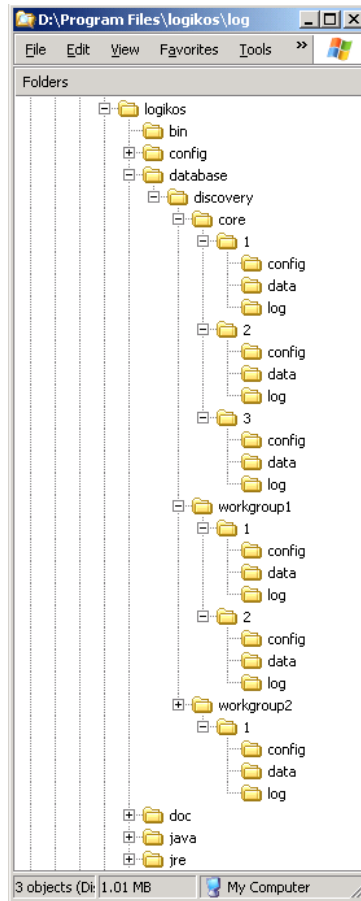
```
NetworkFerret_root/database
```

There are then separate subdirectories for topology (not implemented at this time) and discovery which are created automatically by Network Ferret.

```
NetworkFerret_root/database/
  topology
  discovery
```

Under the topology and discovery subdirectories there will be a directory for each named definition. Using the example in the previous section, the directory tree would look like the following:





config – holds config files used for this discovery
data – holds discovery data
log – holds log files from the discovery run

Working With Discoveries

Normally, all of this work would be done programmatically by the embedding application (see the Programmer's Guide). This section describes two scripts that can be used to evaluate the topology feature without having to write code to embed Network Ferret.

The topology script is used to manipulate the directory tree by creating/deleting definitions or deleting/comparing instances. Empty instances can be created with the topology script as well but it is much more interesting to create discovery instances using the sniffer script. Of course, one can also simply delete directories from the file system rather than using the script. There are no hidden files or links you need to worry about.



The sniffer script will create a discovery instance, run Network Ferret and populate the discovery directory with the data returned by Network Ferret.

Both the topology and sniffer scripts define the name of the topology subdirectory under the Network Ferret root directory. The default directory is called *database*. You can change this if you like but make sure you change it in both scripts.

Creating A Discovery Definition

Run the topology script with the following parameters:

```
topology create discovery definition yourName
```

After the script complete, you should have the following files:

```
NetworkFerret_root/database/discovery/yourName/attributes.txt
```

Attributes.txt contains some information about the discovery definition. If you do not run this command before creating a discovery instance, that is OK. Network Ferret will create the definition with some defaults in attributes.txt.

Creating A Discovery Instance

A discovery instance (version) should be created using the sniffer script. The sniffer script should be run with the following parameters:

```
sniffer configFile definitionName instanceNumber
```

The configFile is the config file used by Network Ferret to run discovery. It is assumed to be in the config directory defined in the logikosroot script. If the definition name does not exist, Network Ferret will create a definition with some default attribution. You should always use a number for the instanceNumber.

Once the sniffer has completed you can see the Network Ferret HTML and CSV files in their normal places. Assuming an instance number of 1 was used, you should also see the following files in the database tree:

```
NetworkFerret_root/database/discovery/yourName/attributes.txt
NetworkFerret_root/database/discovery/yourName/1
NetworkFerret_root/database/discovery/yourName/1/attributes.txt
NetworkFerret_root/database/discovery/yourName/1/config
NetworkFerret_root/database/discovery/yourName/1/config/defaults.cfg
NetworkFerret_root/database/discovery/yourName/1/config/configFileName.cfg
NetworkFerret_root/database/discovery/yourName/1/data
NetworkFerret_root/database/discovery/yourName/1/data/files for each node
```



```
NetworkFerret_root/database/discovery/yourName/1/log
NetworkFerret_root/database/discovery/yourName/1/log/discovery.txt
NetworkFerret_root/database/discovery/yourName/1/log/errors.txt (possibly)
NetworkFerret_root/database/discovery/yourName/1/log/exceptions.txt (possibly)
```

You can also create a discovery instance with no data in it using the topology script as follows:

```
topology create discovery definition yourName
topology create discovery instance yourName versionNumber
```

Deleting Topology Information

The easiest way to delete topology information is to simply delete the directories from the file system. However, the topology script can also be used as follows:

```
topology delete discovery definition yourName
topology delete discovery instance yourName versionNumber
```

Deleting a definition will delete all instances.

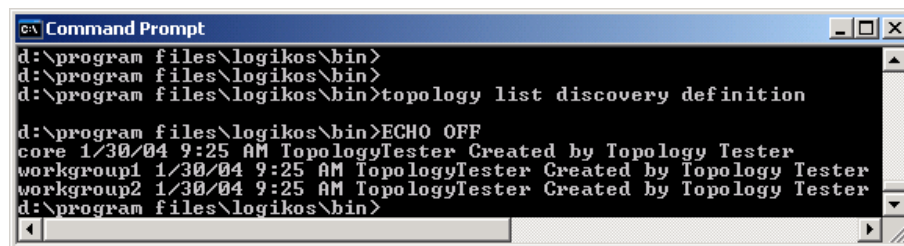
Listing Topology Information

Use the topology script as follows to list all discovery definitions along with their creation date, created by and description information.

```
topology list discovery definition
```

Below is what the output would look like for the example we have been using. The output is

```
Definition name - core
Date Created – 1/30/04 9:25 AM
Created by – TopologyTester (the script)
Description – Created by Topology Tester
```



```
C:\ Command Prompt
d:\program files\logikos\bin>
d:\program files\logikos\bin>
d:\program files\logikos\bin>topology list discovery definition

d:\program files\logikos\bin>ECHO OFF
core 1/30/04 9:25 AM TopologyTester Created by Topology Tester
workgroup1 1/30/04 9:25 AM TopologyTester Created by Topology Tester
workgroup2 1/30/04 9:25 AM TopologyTester Created by Topology Tester
d:\program files\logikos\bin>
```

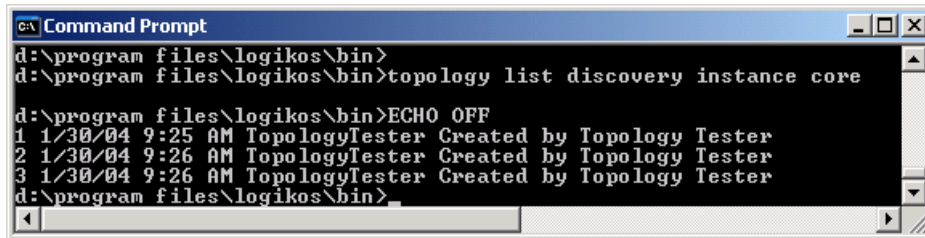
Use the topology script as follows to list all instances of a given discovery definition:

```
topology list discovery instance defintionName
```



Below is what the output would look like for the core definition in the example we have been using. The output is

Version Number - 1
Date Created – 1/30/04 9:25 AM
Created by – TopologyTester (the script)
Description – Created by Topology Tester



```
Command Prompt
d:\program files\logikos\bin>
d:\program files\logikos\bin>topology list discovery instance core

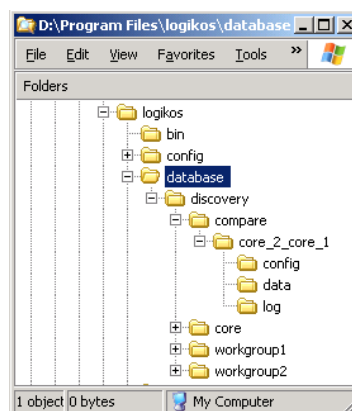
d:\program files\logikos\bin>ECHO OFF
1 1/30/04 9:25 AM TopologyTester Created by Topology Tester
2 1/30/04 9:26 AM TopologyTester Created by Topology Tester
3 1/30/04 9:26 AM TopologyTester Created by Topology Tester
d:\program files\logikos\bin>
```

Comparing Two Discoveries

Networks change over time so it is useful to understand these changes from one discovery to the next. Use the topology script as follows to compare two discovery instances:

```
topology compare discovery instance def2 inst2 def1 inst1
```

The first instance should be the newer one. A subdirectory of discovery will be created which is called compare. The compare directory will have a subdirectory called def1_inst1_def2_inst2. This directory will have empty config and log directories and a data directory containing the comparison data. Below is what the directory structure would look like if we compared versions 2 and 1 of the core discovery.

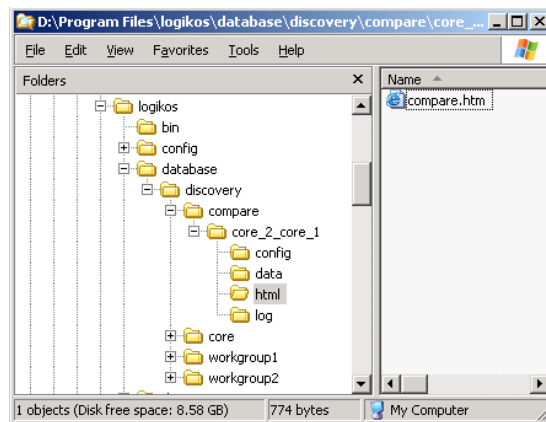


The comparison data is normally expected to be accessed programmatically. To view this data for evaluation purposes run the topology script as follows:

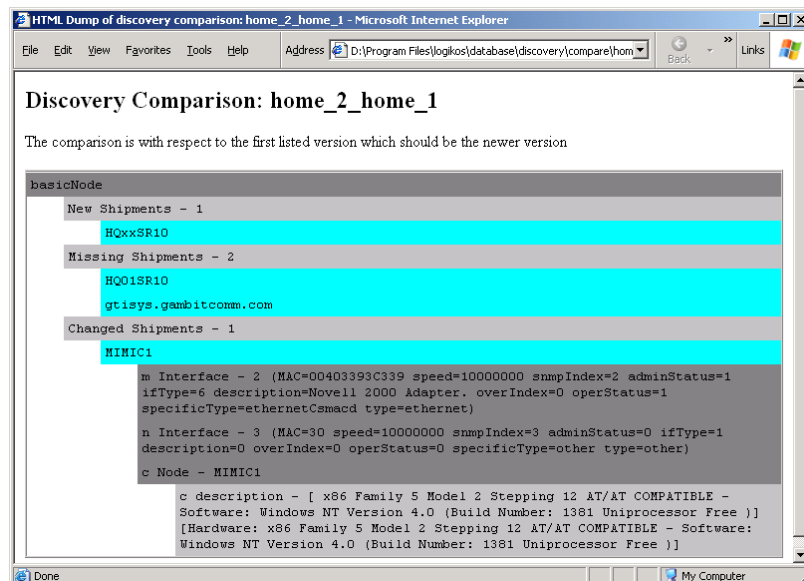
```
topology html discovery instance compare def2_inst2_def1_inst1
```



A directory called html will be created in the comparison instance directory containing one file called compare.htm.



Simply double click to view the comparison results in a browser. Below is a sample result.



The results talk in terms of shipments. Shipments are described in more detail in the Programmers Guide. They are the units of data communicated out by Network Ferret and generally represent a Node. The first line, basicNode, denotes the type of shipment being compared. There are other types as well.

Nodes that are New or Missing will simply list the name of the node. It is assumed that you will have saved the HTML reports from the actual discovery runs to view the complete data. The comparison



data actually contains all of the node information but we wanted to keep the HTML display as compact as possible.

Shipments that are changed will contain indented rows describing the changes. The first level of indentation will indicate either an Entity type (i.e. Board, Interface, etc.) or a Relationship type. An Entity can either be new (n) missing (m) or changed (c). A Relationship can only be either new (n) or missing (m).

In the example above Interface 2 is missing while Interface 3 is new. This is an example of a common occurrence in SNMP. A node will reboot causing a reordering of SNMP indexes. It is up to the embedding application to determine that the missing and new objects are, in fact, the same object but with a different SNMP index.

In the event of a changed Entity, there will be a further level of indentation describing the Attributes of the Entity that have changed. An attribute can also be new, missing or changed. In the case of a new or missing attribute, the value will be given. In the case of a changed attribute, both the old and new values will be given.

In the example above, the Node object has one changed attribute which is the description. Both the original and changed versions of the attribute are given.



Appendix A – MIB Dumps

In order to properly debug a misbehaving device, it may be necessary to take a MIB dump of the device. Network Ferret will take a MIB dump of any address defined in the **IPDDumpAddresses** configuration file parameter.

Note that the MIB Dump Agent is run as part of the core AtiSystem. Network Ferret will pass addresses off to this agent. It is very possible that discovery will finish before all MIB dumps are finished. In this case, discovery will wait until the dumps are complete. The embedding application will receive a progress message saying that discovery is complete, but the callback will not occur until all MIB dumps are complete.



Appendix B – NetConf Dumps

Network Ferret uses NetConf commands to augment SNMP data.

Two VSP parameters allow files to be created of NetConf results.

`ncLogCapabilities`

`ncLogResults`

Setting `ncLogCapabilities` to true will generate a file with a name of the form `vendor_IP_ncCapXML.txt`.

Setting `ncLogResults` to true will generate one or more files with a name of the form `vendor_IP_ncResXML_discoveryFeature.txt`.

Look at the NetConf section of `root.vsp` and you will see that you can use these files in your own internal testing in conjunction with your SNMP simulator.



Appendix C – Messages

All messages are now documented in Javadoc in the `com.logikos.discovery.message.AtStringBundle` class.

