Message Processing

Overview

GroundWork Monitor uses **syslog-ng** to parse incoming syslog messages from remote systems. Syslog-ng is used to suppress uninteresting messages and pass critical messages directly into *Nagios* and into the *GroundWork Foundation* database for alarming and reporting. GroundWork Monitor uses **SNMP Trap Translator** (SNMPTT) to parse incoming SNMP traps and format them in a human readable fashion for display in alarms and the web interface. It also allows for customizing the severities of each trap based on customer requirements. **SNMP polling** allows GroundWork Monitor to query network devices for common operational statistics such as memory usage and network traffic levels, while SNMP traps allow the network devices to unilaterally notify GroundWork Monitor of critical events whenever they occur.

CONTENTS

RELATED RESOURCES

- Syslog-ng
- SNMPTT
- Net-SNMP

WAS THIS PAGE HELPFUL?

- Leave Feedback

1.0 About Message Processing

GroundWork Monitor has two types of messages that are processed by the system:

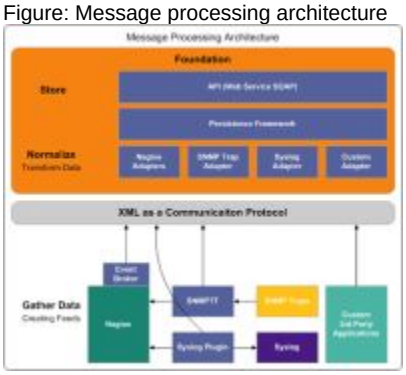
- Status information and events of state changes generated by Nagios and forwarded to Foundation.
- External tools generating events (SNMP and Syslog) which are processed by GroundWork Monitor. External messages are normally persisted in Foundation and in order to generate notifications sent as passive checks to Nagios.

The out of the box GroundWork Monitor is configured to process Nagios events (state change notifications), Nagios status information, SNMP Traps and Syslog messages. The system includes plugins, feeders and Foundation adapters to process these types of data.

The flexibility of the system allows you to define custom feeder/adapters so that any monitoring related data can be integrated by GroundWork Monitor. The diagram below shows a top level overview of the Message Processing Architecture of GroundWork Monitor.

To create a custom feed you will need to 1) Define metadata (properties, application types), 2) Setup runtime system, and 3) Feed monitoring data.

Figure: Message processing architecture



2.0 Syslog Processing

The Syslog processor is a module of open source packages that will enable remote syslog messages to be forwarded to the GroundWork server and post error messages on the operator console when predefined errors occur. The syslog process includes the following components and is displayed in the diagram.

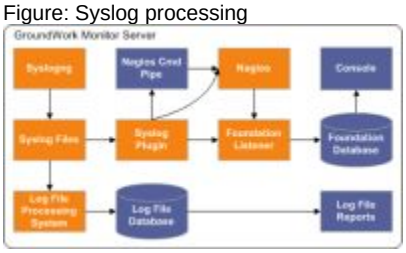
The syslog-ng package is installed on the GroundWork server. It allows remote systems to use the syslog facility to direct log messages to a directory on the GroundWork server. Configuration files will define which log messages are directed to GroundWork and which programs to call

when those messages are found.

A plugin will send event messages to the Foundation LogMessage table. An operator can view a history of log messages by using the Console application. An application type of SYSLOG is predefined in Foundation so the operator may view all SYSLOG messages in the Console. GroundWork's SYSLOG Service Profile which includes Nagios SYSLOG trap plugins executes the following process:

- Reads the designated syslog file on a scheduled basis. Each time the plugin is called, it will read from the last position to the end of the file.
- Compares each line with a match file containing regular expressions. If the regular expression matches, an exception condition is flagged.
- Sends the exception event to be logged in the Foundation database.
- Queries Foundation for number of SYSLOG exception events associated with each Host. It reports the number of exception conditions in the Service output and will set the Service status to the highest severity of the matched message.
- Sends the last matched message to a passive Service named `SYSLOG_last`. This Service can be used for Notification or event handling.
- Configuration definitions for the summary Service and the passive Service check.

Figure: Syslog processing



3.0 SNMP Trap Messages

> **Best Practice**
> The SNMPTrap process is a brittle and difficult to configure subsystem. If too many traps come in, the disk creates enough traps that even stating the spool directory where they are temporarily stored by `snmptrapd` causes a huge delay enough that most processes would either time out or hang. If you get behind, where the rate of incoming traps is greater than the rate you are processing them, problems will show up around disk space reporting, inode availability, disk io, application and network timeouts. You will eventually run out of inodes. Since we turn on this the snmptrap processes at startup but don't have any definitions defined, this is an easy attack vector to take out GroundWork from the network without needing to authenticate. Until a remedy, it is recommended that all systems where snmptraps are not being used (e.g., child servers) have the processes disabled:
> ```
> service groundwork stop snmptrapd
> chmod -x /usr/local/groundwork/common/scripts/ctl-snmptrapd.sh
>
> service groundwork stop snmptt
> chmod -x /usr/local/groundwork/common/scripts/ctl-snmptt.sh
> ```

GroundWork Monitor offers direct support for SNMP polling and SNMP traps.

Support for SNMP Polling

In simple terms, SNMP polling allows GroundWork Monitor to query network devices for common operational statistics such as memory usage and network traffic levels, while SNMP traps allow the network devices to unilaterally notify GroundWork Monitor of critical events whenever they occur.

For example, SNMP polling can be used to read the current disk utilization levels on a device every few hours, with this data being used to monitor or chart long-term consumption trends, and possibly being used to generate an alarm if the usage levels are determined to be abnormal. However, if a critical disk shortage suddenly appears in between these polling intervals, this usage scenario could allow the disk shortage to go undetected for several hours. In that kind of situation, the struggling device would best be served if it could unilaterally transmit an SNMP trap message that indicated the error condition immediately.

How this Works

All SNMP messages are transported by using the User Datagram Protocol (UDP), and this includes SNMP queries and unsolicited trap messages alike. The use of UDP as a transport protocol has several advantages, but it also imposes fairly strict restrictions on the size of the message that can be sent. In order to ensure that an SNMP message will fit within the size constraints, the SNMP protocol uses a variety of encoding and compression techniques that keep the messages relatively small. However, this also means that the messages themselves are binary, and are not readily decipherable with the naked eye.

In particular, SNMP messages do not contain the full text message associated with an event, but instead uses a message ID that is uniquely

associated with the underlying message. Meanwhile, information about the event (such as the severity of the event, and the full-text string message) is stored in a dictionary file called a Management Information Base (MIB), which is independently maintained by the message receiver. This model allows the sending station to simply provide the ID value in the original message, while leaving it to the receiver to map the ID value to an actual event message that can be displayed to a human operator.

For example, an SNMP management station might record an event that says "Link up on interface 2" after a network interface has come online. However, the underlying SNMP message will only contain the fixed identifier value for that event - along with the variable value for the interface that just became active - but does not contain the full "Link up" text message. Instead, the trap sender expects that the recipient system will be able to associate the code value with a local MIB and display the correct message, with the variable data being substituted into the proper location.

Along the same lines, SNMP messages are also able to refer to object IDs outside their own scope, as long as the other IDs are correctly referenced in the dictionary. For example, the numeric value for the "Link up on interface 2" message discussed above uses a numerical syntax that is described in the generic standard messages. As such, the trap dictionary for this event does not need to define the syntax rules for that variable value, but instead can simply reference the parent definition.

Taken as a whole, this approach allows the messages and dictionary files to be very small and efficient, but it also results in making the message data opaque to human users. Therefore, in order for the data to be usable by a human, this model requires that the SNMP management station maintain a dictionary of trap messages and their ID assignments, with the incoming messages being mapped to their full-text event message, with the message variable data being decoded and applied where needed.

The SNMP Trap Software Architecture

By default, GroundWork Monitor provide a basic SNMP trap handler that captures a handful of common messages, which are then stored in the GroundWork Foundation database for viewing through various applications, and which are also fed to Nagios for the purpose of triggering notification messages.

More specifically, GroundWork Monitor uses the following components to receive, process, and store SNMP trap messages as they arrive from the network:

- Incoming SNMP traps are received by the `snmptrapd` daemon that is provided with the Net-SNMP software package. `snmptrapd` provides the network listener that is needed to receive incoming SNMP trap messages from the network.
- Once a trap message has been received, `snmptrapd` passes the incoming message to the snmptthandler program that is part of the SNMP Trap Translator software package, which in turn dumps the trap message into a spool file. This allows `snmptrapd` to dispose of the messages quickly.
- Periodically, the snmptt daemon included with the SNMP Trap Translator package examines the spool file for new trap messages, converts them into structured text, and then simultaneously passes the output to the GroundWork Foundation database and the Nagios NSCA passive interface. This conversion process allows the SNMP messages to be processed the same as other kinds of textual event messages.
- Trap messages that are received by the Foundation database listener are correlated with the original host device and then recorded as SNMPTRAP event messages. Once the database has been updated, the trap data can be displayed by the operator using the global Event Viewer or the host-specific Status applets.
- Meanwhile, trap messages that are received by Nagios through the NSCA interface are stored in the Nagios command file until the next check interval. When these passive checks are subsequently processed by Nagios, the incoming messages will be correlated with the original host device and processed as events for the `snmptraps_last` service associated with that host. From there, the messages can then be used to perform whatever actions are available in Nagios, such as generating notification messages.

Taken as a whole, this design allows GroundWork Monitor to perform as a comprehensive SNMP monitoring and management station, once the individual software components have been configured.

Prior versions of GroundWork Monitor Professional used a somewhat different architecture with different software components. If the SNMP trap support from a previous version of GroundWork Monitor is already in use, the older configuration files will need to be converted before those customizations will perform correctly with version 5.2 and beyond. GroundWork Monitor provides a conversion script in `/usr/local/groundwork/tools/snmp_mibs/upgrade_snmptt_conf.sh` that will perform this process somewhat automatically.

The SNMP Trap Listener

GroundWork Monitor incorporates the full Net-SNMP software package, including the `snmptrapd` daemon that monitors for incoming trap messages on UDP port 162. `snmptrapd` is enabled by default, and will be automatically started after installation.

GroundWork Monitor does not provide a dedicated startup script for `snmptrapd`, but instead starts the daemon as part of the `/etc/init.d/snmptt startup` script that is used to initialize the SNMPTT service. If you ever need to restart the `snmptrapd` daemon, you should use that control script.

Only one SNMP listener can be active on the default port for a network interface, so if you are already using another trap listener for some other purpose you will need to either configure the listeners to use separate network interfaces or port numbers, or integrate the configuration files together (if you are already using `snmptrapd` from another source, then you may be able to simply merge the configuration files).

The configuration settings for the `snmptrapd` daemon bundled with GroundWork Monitor are stored in `/usr/local/groundwork/common/etc/snmp/snmptrapd.conf`. The default configuration file contains the minimum directives that are needed to capture SNMP traps and pass them to `snmptthandler` for processing, but additional configuration directives may also be added if they are needed.

The default configuration file does not define any filters, so all incoming trap messages will be intercepted in the default setup. If you need to limit SNMP trap handling to specific hosts or a particular SNMP community, you will need to edit this configuration file accordingly. Otherwise, you can simply instruct your network devices to begin sending SNMP traps to the GroundWork Monitor server, and the messages will immediately be received and processed without any local configuration required.

The SNMP Trap Translator

The raw form of an SNMP trap message uses an opaque binary syntax that is not directly usable by GroundWork Monitor or Nagios. For that reason, GroundWork Monitor relies on the open source SNMP Trap Translator (SNMPTT) package to convert the raw SNMP traps into structured textual messages that can be manipulated more easily.

Specifically, `snmptrapd` passes incoming SNMP traps to a program called `snmptthandler`, which in turn stores the trap messages into an incoming spool file. Every few seconds, an independent `snmptt` daemon reads the spool file for new trap messages, converts the messages into structured text, and then passes the output to the GroundWork Foundation database and the Nagios NSCA agent through the appropriate local interfaces.

SNMPTT uses multiple configuration files to determine how trap messages should be converted and relayed. First and foremost, `/usr/local/groundwork/common/etc/snmp/snmptt.ini` contains the global options that are needed for the `snmptt` daemon to operate as expected (such as database connection and logging options), and also contains pointers to subordinate trap-specific configuration files that tell the `snmptt` daemon how to dispose of specific trap messages.

The actual translation and disposal process is controlled by trap-specific configuration files. For example, `/usr/local/groundwork/common/etc/snmp/snmptt.conf` contains the processing directives for a handful of universal SNMP trap messages, and GroundWork Monitor also includes predefined translation files for a few specific devices which are typically stored in their own configuration files (the `CISCO-GENERAL-TRAPS.conf` configuration file contains mapping information for SNMP traps that are commonly found with Cisco network equipment, and so forth). Each of these translation-specific configuration files are already listed in the `snmptt.ini` control file, and will be used to process the corresponding SNMP trap messages.

If you need to capture the full trap message and severity details for SNMP traps that are not already defined, you must extract this information from the appropriate SNMP MIB files and add the trap messages to one of the control files (along with information that tells the `snmptt` daemon where to send the messages). GroundWork Monitor provides a variety of tools to simplify and automate portions of this process, although some manual intervention is also usually required. Refer to the Adding New Trap Definitions section below for information on how to add new trap translation definitions.

GroundWork Foundation SNMP Trap Adapter

When the `snmptt` daemon needs to pass a converted trap message to the GroundWork Foundation event database, it passes the output to the Foundation network listener on TCP port 4913 on the loopback interface (127.0.0.1), and flags the data as an SNMPTRAP event with an appropriate severity level. In turn, the SNMPTRAP Foundation adapter examines the data and adds the event to the GroundWork Foundation LogMessage database table.

SNMPTT is configured by default to send all trap messages to the GroundWork Foundation listener, including unknown SNMP trap messages that have not been listed in `snmptt.ini` or a subordinate configuration file. These messages will be shown with an "unknown" severity level, and will show "unknown trap" in the message text. Refer to the Adding New Traps section below for information on adding new trap definitions to the `snmptt` configuration.

By default, the SNMPTRAP adapter will consolidate duplicate event messages that are received in sequence. If a trap event arrives that matches the consolidation criteria, the duplicate event message is discarded, the "count" field of the existing record is incremented, and the "last insert date" field is updated. Specifically, the consolidation feature looks at the following fields to determine if a record is a duplicate, and if all fields match then the duplicate is discarded:

- OperationStatus
- Host
- Severity
- IP Address
- MonitorStatus
- Event_OID_numeric
- Event_Name
- Category
- Variable_Bindings

An exception to the above is if the Monitoring Status of the last event is different than the Monitoring Status of the incoming event. In that scenario, a new event message will be created. This rule guarantees that console messages sorted by chronological order will always show the current status above previous status.

You can disable trap consolidation if you wish (thereby having a new event record be created for every SNMP trap). You can also change the fields that are used for the consolidation service if needed. The fields that are used for the consolidation function are stored in the SNMPTRAP entry of the ConsolidationCriteria table of the GWCollageDB database.

Once the event records have been inserted into the event database, the trap messages can be viewed from the Event Console application (selecting the SNMPTRAP option in the left navigation tree will display only SNMP trap events, and filter out other messages). Recent SNMP trap messages that have been correlated with a host can also be viewed in the Status application by selecting that host and looking under the "Console" portion of the host status screen.

The Nagios SNMP Trap Handler

The `snmptt` daemon also passes incoming trap data to Nagios via the NSCA passive interface on TCP port 5667 on the loopback interface (127.0.0.1), and flags the data as a passive check result for the `snmptraps_last` service on the originating host. From there, Nagios will periodically process the incoming passive check queue and attempt to correlate the incoming events with the `snmptraps_last` service definition associated with the originating host.

More specifically, the `snmptt` daemon provides the translated trap message, and also provides the Nagios-specific result codes that indicate whether the trap is benign, a warning, a critical event, or an unknown message. In turn, Nagios is able to use the trap severity to determine when to initiate notifications and perform other actions, and can also display the trap text in the service status field.

> ⚠ SNMPTT is configured by default to send all trap messages to the NSCA listener, including unknown SNMP trap messages that have not been listed in `snmptt.ini` or a subordinate configuration file. These messages will be shown with an "unknown" severity level, and will show "unknown trap" in the message text. Refer to the Adding New Traps section below for information on adding new trap definitions to the `snmptt` configuration.

In order for this process to work, the Nagios NSCA passive interface must be active, the originating host must be defined in Nagios, and the originating host entry must have the `snmptraps_last` service definition associated with it.

> ⚠ The Nagios NSCA passive check interface is not active by default, and must be started before the SNMP trap messages can be received. The `/etc/init.d/nsca init` script is provided for this purpose. This init script should be included with the system init scripts if the NSCA service will always be needed.

> ⚠ The `snmptraps_last` service definition is not included in the default set of Nagios service definitions, although the associated service profile is provided with GroundWork Monitor.

To use this service definition, perform the following steps:

1. Select the Configuration application from the main GroundWork Monitor menu.
2. Select Profiles.
3. Select Profile Importer. This will result in a list of all of the available host and service profiles being displayed in the main window.
4. Locate the entry for `service-profile-snmp-traps.xml` in the list of available profiles, and activate the checkbox next to it.
5. Scroll to the bottom of the main window, and click the Import button. This will result in the `snmptraps_last` service definition being installed into the current configuration.

Once the above steps have been completed, the `snmptraps_last` service definition will be available for use, and can be associated with any defined host.

Adding New Trap Definitions

As was discussed earlier, the SNMP Trap Translator toolkit is responsible for converting the incoming binary trap messages into structured text, and for passing the decoded messages to the Foundation database and the Nagios NSCA passive interface. When this has been properly configured, the `snmptt` daemon is able to generate the correct fully-formed textual message, and can also associate an appropriate severity level for a trap message. However, the SNMP Trap Translator is only preconfigured for a few common trap messages, and in all other cases the `snmptt` daemon will not be able to provide an appropriate severity level, nor will it be able to provide a meaningful status text message (it will use a generic "unknown" trap message instead).

If you need to capture the full trap message and severity details for SNMP traps that are not already defined, you must extract this information from the appropriate SNMP MIB files and add the trap messages to one of the control files (along with information that tells the `snmptt` daemon where to send the messages). GroundWork Monitor provides a variety of tools to simplify and automate portions of this process, although some manual intervention is also usually required.

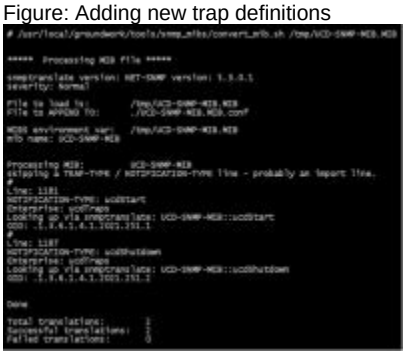
In particular, the `/usr/local/groundwork/tools/snmp_mibs/convert_mib.sh` script will read through a specified MIB file and locate any trap definitions contained therein. If there are no problems with the MIB file (such as unsatisfied dependencies or syntax errors), the convert_mib.sh script will create a local .conf file for use with `snmptt` that contains all of the trap definitions and the options that are needed for

snmptt to correctly process and relay those trap messages. From there, the output .conf file can be added to the snmptt.ini control file, and the new trap messages will be processed after the snmptt daemon has been reinitialized. However, if the original MIB file cannot be converted (possibly due to unsatisfied dependencies or syntax errors), additional steps will need to be taken before the SNMP traps can be captured.

1. The first step in this process is to locate the SNMP MIB file that contains the trap messages you want to capture and convert. Most equipment vendors provide the necessary MIB files with their installation software. There are also a variety of third-party MIB repositories across the Internet that can be helpful for locating MIB files.
2. Once the MIB file has been obtained, use the /usr/local/groundwork/tools/snmp_mibs/convert_mib.sh script to automatically convert it into a snmptt configuration file by specifying the MIB file name as the only parameter. If the conversion operation is successful, a new file will be created in the local directory with same name as the input file, with an extra .conf suffix at the end.

   For example, the figure below shows the command convert_mib.sh /tmp/UCD-SNMP-MIB.MIB. This command causes the convert_mib.sh script to read in the /tmp/UCD-SNMP-MIB.MIB file, locate the SNMP trap definitions in the MIB file, and then generate a SNMP Trap Translator configuration file called UCD-SNMP-MIB.MIB.conf in the current directory. In that example, the output file contains definitions for two SNMP traps, which are ucdStart and ucdShutdown, respectively.

   Figure: Adding new trap definitions

   > ⚠ The conversion script will append data to the output file instead of overwriting it. If you want to completely regenerate an existing configuration file, you must delete the existing configuration file before running the script again.

3. Once the desired configuration file has been created, it must be added to the /usr/local/groundwork/common/etc/snmp/snmptt.ini control file before it will be used to interpret matching traps. To add the new configuration file to the control file, open snmptt.ini in a text editor, locate the "TrapFiles" section towards the bottom of the file, and then add the full path and filename of the new configuration file somewhere before the "END" directive. To simplify management, it is suggested that the new configuration file be copied to the /usr/local/groundwork/common/etc/snmp/ directory beforehand.
4. After the snmptt.ini file has been updated, restart the snmptt daemon with the following command:

   ```
   $ /etc/init.d/snmptt restart
   ```

   Once the above steps have been completed, the SNMP trap translator will begin processing the newly-defined SNMP traps, and will begin generating the correct text strings and severity levels to GroundWork Foundation and Nagios. However, in those cases where the MIB file could not be converted, some amount of manual intervention will be required.

Resolving MIB Dependencies

Almost all MIB files are somewhat dependent on other MIB files for certain types of data. For example, an SNMP trap may reuse a universal data-type that is defined in another MIB file, and in that scenario the trap definition will not be usable until the other MIBs have also been made available to the MIB conversion utility.

GroundWork Monitor includes several common dependency MIBs already, and in many cases the dependency issue will not be noticed. For example, the screen capture in the preceding section shows the convert_mib.sh script successfully processing the UCD-SNMP-MIB.MIB file, even though the MIB file has several dependencies. However, this is only possible because GroundWork Monitor already includes the dependency MIBs and which are therefore already available to the conversion utility.

If the conversion script fails due to one or more missing dependency MIBs, you will need to determine which MIB files are missing, and then install them in the appropriate location. This can be accomplished in the following way.

You can also review the MIB file by opening it in a text editor. The dependencies will be shown as lines of text beginning with the word "IMPORTS", followed by a list of objects to be imported, then a "FROM" clause followed by the name of the source MIB. For example, the following block of text from the MIB file for a Dell PowerConnect 3248 switch, and refers to multiple subordinate MIBs:

```
IMPORTS
        MODULE-IDENTITY, OBJECT-TYPE, Unsigned32, Integer32, IpAddress,
        NOTIFICATION-TYPE, internet, Counter32
                FROM SNMPv2-SMI
        DisplayString, RowStatus, TruthValue
                FROM SNMPv2-TC
        EnabledStatus
                FROM P-BRIDGE-MIB
        BridgeId, Timeout, dot1dStpPortEntry, dot1dStpPort
                FROM BRIDGE-MIB
        PortList
                FROM Q-BRIDGE-MIB
        ifIndex
                FROM IF-MIB;
```

In the example above, the MIB file has dependencies on SNMPv2-SMI, SNMPv2-TC, P-BRIDGE-MIB, BRIDGE-MIB, Q-BRIDGE-MIB, and IF-MIB. Each of these MIB files will be needed before the SNMP trap messages in the original MIB can be converted for use by `snmptt`.

Once you have determined the dependency MIBs, you must locate the appropriate MIB files and add them to the MIB repository in GroundWork Monitor.

For clarification purposes, it is important to understand that the `convert_mib.sh` script is a front-end to the `/usr/local/groundwork/common/bin/snmpttconvertmib` Perl script that is provided with the SNMP Trap Translator package, which ensures that the appropriate options and command-line parameters are used when the configuration file is generated. However, the `snmpttconvertmib` script uses the `/usr/local/groundwork/common/bin/snmptranslate` utility that is provided with the Net-SNMP software package for the task of decoding the canonical MIB data. As such, the Net-SNMP software package is ultimately responsible for locating the dependency MIBs when the traps are first converted to text.

All of the Net-SNMP utilities use a common cache directory for common MIB files, which is the `/usr/local/groundwork/common/share/snmp/mibs` directory. Any MIB file that is stored in this directory is available to all of the Net-SNMP utilities for the purpose of converting and displaying MIB data in text form, and this includes the one-time conversion that is utilized whenever a `snmptt` configuration file is generated.

Therefore, in order for a dependency MIB to be recognized by the conversion script, the dependency MIB file must be added to the `/usr/local/groundwork/common/share/snmp/mibs` directory. Once the file has been copied to that directory (and the appropriate file permissions have been defined), the conversion script can be called again, and the dependent MIB files will be located and automatically used when the `snmptranslate` utility is ultimately called upon to generate the trap text.

By default, the `/usr/local/groundwork/common/share/snmp/mibs` directory contains several dozen common dependency MIB files. In addition, GroundWork Monitor also includes several other common MIB files under the `/usr/local/groundwork/common/share/mibs` directory tree (note that this is a different directory than the Net-SNMP cache directory), and those MIB files can be copied into the Net-SNMP cache directory as they are needed. If a MIB file has additional dependencies that are not provided by GroundWork Monitor, you may need to locate the dependency MIB files on the equipment vendor's support site or on an Internet MIB repository site.

Using the Dell PowerConnect 3248 MIB example described above, we can tell that the MIB file has dependencies on SNMPv2-SMI, SNMPv2-TC, BRIDGE-MIB, P-BRIDGE-MIB, Q-BRIDGE-MIB, and IF-MIB. Furthermore, the Net-SNMP cache directory already contains the MIB files for SNMPv2-SMI, SNMPv2-TC, and IF-MIB, but does not contain the BRIDGE-MIB, P-BRIDGE-MIB or Q-BRIDGE-MIB MIB files. However, those files do exist in the `/usr/local/groundwork/common/share/mibs/ietf` repository. As such, they only need to be copied into the `/usr/local/groundwork/common/share/snmp/mibs` directory, at which point they will be available to the `snmptranslate` utility whenever the `convert_mib.sh` script is next executed.

Note that there can sometimes be multiple nested levels of dependencies, and it may be necessary to copy more MIB files into the Net-SNMP cache directory than originally expected. For example, Q-BRIDGE-MIB has a dependency on RMON2-MIB, which in turn has a dependency on TOKEN-RING-RMON-MIB. In order for the PowerConnect 3248 MIB file described above to be processed without errors, all of these MIB files would need to be added to the Net-SNMP cache directory.

Verifying MIB Syntax

Many SNMP MIB files contain a variety of errors, some of which are relatively benign, while others are sufficiently problematic that `snmptt` will be unable to determine how to translate the MIB. In the latter case, this may require you to locate a different version of the MIB file, or to perform some editing of the file itself (this should not be done except in extreme cases - SNMP MIB definitions use a very complicated notation format, and it is very easy to make a bad situation much worse).