Contents

Preface
Intended Audience .......................................................... v
Text Conventions .............................................................. v
Documentation Feedback.................................................... vi
Online Documentation .......................................................... vi

Chapter 1: Introduction

Chapter 2: The GnSNMPDev Management Module
Modeling ........................................................................... 3
Identification ....................................................................... 4
Interfaces ........................................................................... 4
Applications ....................................................................... 5
Views ................................................................................... 6
Traps, Events, and Alarms ....................................................... 6

Chapter 3: Customizing the GnSNMPDev Management Module
Adding Support for Additional MIBs ...................................... 7
Adding Support for Additional Traps ........................................ 8
Adding Watches to Monitor and Manage Model Conditions .......... 9
Specifying Device Type Names for Device Identification ............... 9

Chapter 4: Developing a New Management Module
Choosing to Develop a New Management Module ..................... 11
  Additional MIB Support ....................................................... 12
  Unique Trap Mapping ......................................................... 12
  Unique Watches ............................................................... 13
  Interface Model Creation .................................................... 13
Creating a New Device Model Type ......................................... 14
  Designing a New Device Model Type ..................................... 14
Creating a New Device Model Type ......................................... 15
  Configuring a New Device Model Type ................................. 15
    Setting Model Type Flags .................................................. 16
    Setting Attribute Values ................................................... 16
    Mapping a Device or a Device Family to the New Device Model Type 20
    Configuring Serial Number Handling .................................. 25
Creating a New Application Model Type .................................... 25
  Understanding Derivation Points and Model Fragments ............. 25
Choosing a Derivation Point .................................................. 27
Board and Port Considerations .............................................. 27
Creating Application Model Types ......................................... 30
Index
Preface

Welcome to the *Generic SNMP Device Management User Guide and Toolkit (1316).*

**Intended Audience**

This guide is intended to explain the function of SPECTRUM’s generic management module, GnSNMPDev, at three different levels:

- As a generic management module that can be used to represent an SNMP-compliant network device that does not have a corresponding SPECTRUM management module
- As a customizable management module to which you can add new watches, device type information, or additional MIB support
- As a toolkit to create new management modules to support devices that are not SNMP-compliant or require complex customizations

**Text Conventions**

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Element</th>
<th>Convention Used</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td><em>Courier and Italic in angle brackets (&lt;&gt;)</em></td>
<td>Type the following: DISPLAY=&lt;workstation name&gt;:0.0 export display</td>
</tr>
<tr>
<td>(The user supplies a value for the variable.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The directory where you installed SPECTRUM</td>
<td><em>&lt;$SPECROOT&gt;</em></td>
<td>Navigate to: &lt;$SPECROOT&gt;/app-defaults</td>
</tr>
<tr>
<td>(The user supplies a value for the variable.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solaris and Windows directory paths

Unless otherwise noted, directory paths are common to both operating systems, with the exception that slashes (/) should be used in Solaris paths, and backslashes (\) should be used in Windows paths.

Example:

<$SPECROOT>/app-defaults on Solaris is equivalent to <$SPECROOT>\app-defaults on Windows.

On-screen text

Courier

The following line displays:

path=’/audit’

User-typed text

Courier

Type the following path name:

C:\ABC\lib\db

Documentation Feedback

To send feedback regarding SPECTRUM documentation, access the following web address:

http://supportconnectw.ca.com/public/ca_common_docs/docserver_email.asp

Thank you for helping us improve our documentation.

Online Documentation

SPECTRUM documentation is available online at the following web address:

http://ca.com/support

Check this site for the latest updates and additions.
Chapter 1: Introduction

SPECTRUM provides a generic management module that can be used to represent an SNMP-compliant network device that does not have a corresponding SPECTRUM management module. SNMP-compliant devices are supported by a series of Management Information Bases (MIBs), which are SNMP structures that describe the particular device being monitored. MIBs are imported into the SPECTRUM database and made available via device, application, and interface model types.

The generic model type, GnSNMPDev, is able to efficiently represent a broad range of devices by creating the following:

- A single model to represent the device
- Application models to represent each of the standard (IETF) MIBs that are supported by the device
- Interface models to represent the device’s ports

In many cases, the functionality provided by the GnSNMPDev management module is adequate to properly manage the device. However, there are cases where you may need to extend the capabilities of the GnSNMPDev management module to support additional MIBs or represent specialized features.

This manual provide instruction on the following:

- Using the GnSNMPDev management module—You can use the GnSNMPDev management module to manage a standards-compliant device for which a specific management module is not available.
- Customizing the GnSNMPDev management module—You can enhance the GnSNMPDev model type in several ways, for example, by importing MIB objects from standard or proprietary MIBs or adding additional processing for traps, events, and alarms.
- Developing a new management module—You can use the GnSNMPDev management module as a toolkit to derive new device model types and application model types. Once the new model types have been derived, you can configure how they will process traps.
Chapter 2: The GnSNMPDev Management Module

The GnSNMPDev management module provides management for standards-compliant, SNMP devices for which a specific management module is not available. GnSNMPDev is an extremely powerful modeling capability because it allows SPECTRUM to dynamically create models on-the-fly to manage devices when a specific management module is unavailable.

GnSNMPDev rapidly queries the device to determine its characteristics and capabilities and then creates a model to represent the device. GnSNMPDev also creates the following:

- Sub-models, referred to as application models, to represent each of the standard MIBs supported by the device
- Interface models to represent each device port defined in the standard MIB-II interface table

The application and interface models are associated with the GnSNMPDev device model, and together they provide the basic management capabilities for the device.

Devices that are modeled with the GnSNMPDev model type can be used with all of SPECTRUM’s management tools. Importantly, GnSNMPDev models participate fully in SPECTRUM’s root cause analysis, fault isolation, and downstream alarm suppression algorithms. As a result, they are able to alert users to network and device problems like other SPECTRUM device models.

Modeling

When modeling a device using Discovery or the Model by IP Address icon, SPECTRUM automatically chooses the GnSNMPDev model type when a specific management module for the device is not available. You can also choose to model a device using the GnSNMPDev model type when using the Model by Type icon to model a device.

As with all management modules, you can map the connectivity of interface models automatically using Discovery, or you can map it manually.
The GnSNMPDev model type supports the Cisco Proprietary Discovery Protocol (CDP). A CiscoCDPApp application model (see Applications on page 5) will be created for Cisco devices modeled with GnSNMPDev that support CDP. This application model allows SPECTRUM to use the device’s Proprietary Discovery tables when discovering connectivity information for the device.

For more information on modeling your network, see the *Modeling Your IT Infrastructure Administrator Guide (5167)*.

### Identification

When a device is modeled with GnSNMPDev, SPECTRUM will assign the device a descriptive identifier or device type, and an icon shape and label that reflects the device's functionality. See the *Modeling Your IT Infrastructure Administrator Guide (5167)* for more information on the icon shapes and labels used in SPECTRUM.

To determine the model's device type, SPECTRUM will first check the System Object Identifier-to-Device Type mapping list (see Specifying Device Type Names for Device Identification on page 9). If a device type name (for example, "Cisco 2621") is found for the device's System Object Identifier, it becomes the model's device type. If no match is found, SPECTRUM extracts the device's enterprise ID from the System Object Identifier and uses it to determine the manufacturer. SPECTRUM then looks at the device’s capabilities and—based on these—appends an abbreviation (for example, Rtr or Bdg) to the manufacturer name. This entire string becomes the device type name (for example, "Cisco Rtr"). If SPECTRUM is unable to determine an appropriate device type, the default value “SNMP DV” is assigned.

When determining the appropriate icon shape and label for a device model, SPECTRUM determines the primary function of the device (router, bridge, switch, workstation, and so on), and assigns an icon shape and label to the device model that represents this primary function. This icon of the device model appears throughout OneClick.

### Interfaces

GnSNMPDev intelligence creates an interface model for every instance in the MIB-II Interface table. Interface models are instantiated and associated with the device during SPECTRUM modeling. They represent the physical or logical connections on a device.

The device model’s Interfaces tab in the Component Details pane shows all of the interfaces that SPECTRUM has discovered on a device. The view shows the interface’s status (UP or DOWN), among other information.
Connectivity between devices can be mapped to the port level, which gives SPECTRUM the ability to isolate faults to the same level. For example, if a port on a device goes down, an alarm will be generated on the individual interface model rather than at the device level. Interface model statistics can be polled and logged, allowing you to monitor and manage the device’s performance to the interface level.

Potential interface model types include the following:
- Gen_If_Port
- Serial_If_Port
- VLAN_IF
- FrameRelayPort

If Frame Relay Manager is installed and the device supports either of the Frame Relay standard MIBs—RFC1315 or RFC2115—the DLCI circuits will be modeled using the DLCI_port model type. See the Standards-Based Protocol Reference Guide (5186) for more information.

If ATM Circuit Manager is installed and the device supports the ATM MIB RFC1695, the ATM logical connections will be modeled using the ATMVclLink or ATMVplLink model types. See the ATM Circuit Manager User Guide (3518) for more information.

Applications

When a device is modeled with GnSNMPDev, SPECTRUM creates application models to represent each of the standard (IETF) MIBs that the device supports. Application models are instantiated and associated with the device during SPECTRUM modeling.

For example, if GnSNMPDev intelligence detects that a modeled device performs routing functions (based on the presence of a routing MIB), a Routing Application model will be created and associated with the device model. In this manner, non-routing devices are not burdened with the functionality and attributes needed to manage routers; each device model carries only the functionality it needs.

Additional support for standard or proprietary MIBs can be added to the GnSNMPDev model type by customizing the GnSNMPDev management module.

To search for and access the application models associated with a given device model, use the Locater tab in OneClick. There are several predefined searches for application models, but you can also perform a search using custom criteria.
Note: For information about standard MIB applications and accessing their views in OneClick, see the Standards-Based Protocol Reference Guide (5186) and the Host System Resources Management User Guide (5179). For information on using the Locater tab, see the OneClick Console User Guide (5130).

Views

A device modeled with the GnSNMPDev model type offers access to all of OneClick’s views, such as Information, Performance, and Alarms. See the OneClick Console User Guide (5130) for more information on these views.

Traps, Events, and Alarms

The trap support available with the GnSNMPDev management module for the 6 generic traps is shown in the following table.

<table>
<thead>
<tr>
<th>Trap Name</th>
<th>OID</th>
<th>Variable Binding</th>
<th>Event Generated</th>
<th>Alarm Generated</th>
<th>Alarm Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>coldStart</td>
<td>0.0</td>
<td>NA</td>
<td>0x10306</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>warmStart</td>
<td>1.0</td>
<td>NA</td>
<td>0x10307</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>linkDown</td>
<td>3.0</td>
<td>1.3.6.1.2.1.2.1.2.1.1 1.3.6.1.2.1.2.2.1.3 1.3.6.1.2.1.2.2.1.7 1.3.6.1.2.1.2.2.1.8</td>
<td>0x220002</td>
<td>0x220001</td>
<td>yellow alarm on the device (can be configured per port); red alarm on the port</td>
</tr>
<tr>
<td>linkUp</td>
<td>2.0</td>
<td>1.3.6.1.2.1.2.1.2.1.1 1.3.6.1.2.1.2.2.1.2 1.3.6.1.2.1.2.1.3 1.3.6.1.2.1.2.2.1.7 1.3.6.1.2.1.2.2.1.8</td>
<td>0x220001</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>authenticationFailure</td>
<td>4.0</td>
<td>NA</td>
<td>0x1030a</td>
<td>0x1030a</td>
<td>yellow</td>
</tr>
<tr>
<td>egpNeighborLoss</td>
<td>5.0</td>
<td>1.3.6.1.2.1.8.5.1.2</td>
<td>0x1030b</td>
<td>0x1030b</td>
<td>yellow</td>
</tr>
</tbody>
</table>

In addition, the GnSNMPDev model type supports various RFC and IEEE standard applications traps and any traps defined at the global level. For more information on global traps, see the Modeling Your IT Infrastructure Administrator Guide (5167).

It is possible to enhance this support to include other traps and event processing. See Adding Support for Additional Traps on page 8 for more information.
Chapter 3: Customizing the GnSNMPDev
Management Module

If you need to extend the capabilities of the GnSNMPDev management module in order to support a specialized device, you can do so in several ways:

- Add support for additional MIBs
- Add support for additional alerts, events, and alarms
- Add watches to monitor and manage model conditions
- Assign device type names to provide specific device model identification

Adding Support for Additional MIBs

You can add support for a standard or proprietary MIB using the MIB Tools application in OneClick. The high-level process is as follows:

1. Identify the MIB that contains the desired MIB objects.
2. In MIB Tools, import the MIB into the MIB Tools database.
3. Also in MIB Tools, add support in SPECTRUM for the MIB objects by creating corresponding attributes in the SpectroSERVER database.
4. Display the MIB objects in OneClick by extending the Information view of the device. You can do this by modifying an existing subview or creating a new subview.

Note: For more information on using MIB Tools, see the Modeling Your IT Infrastructure Administrator Guide (5167). For more information on creating subviews, see the OneClick Customization Guide (5160).

It is also possible to create a new application model type to support the functionality of a new MIB. You will want to do this if you need to create proprietary port and board models. For information on creating these, see Modeling Ports and Boards on page 34. For general information on creating an application model type, see Creating a New Application Model Type on page 25.
SPECTRUM is able to notify you about significant occurrences on your network through the use of traps (alerts sent from SNMP-compliant devices), events, and alarms.

An alert is an unsolicited message sent out by a managed node on a network. A more specific definition of an alert depends on the management protocol that is used to report the alert. In general, SPECTRUM uses SNMP as the management protocol to communicate with devices on a network. Alerts that are generated by an SNMP-compliant device are called traps. Traps are received by SPECTRUM and converted to events for further processing.

An event is an object in SPECTRUM that indicates that something significant has occurred within SPECTRUM itself or within the managed environment. Events always occur in relation to a model. When a managed element on the network generates an alert, this alert is mapped to a SPECTRUM event in the appropriate AlertMap file. The event is then generated and takes on the event code specified in the AlertMap file.

An alarm is an indication that a user-actionable abnormal condition exists on a model. A model usually detects an abnormal condition when an event occurs, and the EventDisp file indicates that an alarm should be generated.

By default, the GnSNMPDev model type supports various traps, events, and alarms. For a complete listing of this support, refer to Traps, Events, and Alarms on page 6.

You can also add support for additional traps using the MIB Tools application and the Event Configuration application in OneClick. The high-level process is as follows:

1. Identify the MIB that contains the desired trap definitions.
2. In MIB Tools, import the MIB into the MIB Tools database.
3. Also in MIB Tools, map the traps to events and specify the events that should generate alarms (and the severity of the alarms).

When you complete this step using MIB Tools, MIB Tools automatically creates and installs the appropriate event and alarm support files.

4. Launch Event Configuration directly from MIB Tools:
   a. In the Trap Support table, select the mapped traps for which you want to configure events and alarms.

   b. Click (Edit traps for selected items in the trap support table).

5. In Event Configuration, complete the configuration of the events and alarms.

For example, you should specify the symptoms, probable causes, and recommended actions for the alarms. These messages are displayed in OneClick when the alarms are generated.

You might also want to specify additional event processing for one or more events, such as logging the event or using the event to clear an alarm or generate another event using event rules.
In addition, you can customize the default event message that is displayed in OneClick when the events are generated.

**Note:** For more information on using MIB Tools, see the *Modeling Your IT Infrastructure Administrator Guide* (5167). For more information on configuring events and alarms, see the *Event Configuration User Guide* (5188).

### Adding Watches to Monitor and Manage Model Conditions

You can use watches to create one or more watches for a particular model. A watch monitors and analyzes changing internal and external attribute values of a model. Watches can be built to include expressions that incorporate one or more attribute values. These attribute values, or an expression based on these values, can then be measured against a defined threshold value. SPECTRUM evaluates the attribute values defined in a watch by polling the attributes when they are updated or polling them when the watch value is read. Results can be used to generate events and alarms and can be logged for historical tracking and report information. Results can also be sent to script files.

Keep in mind that watches can have an impact on network traffic and system resources. Watches that are no longer useful should be deleted.

**Note:** For more information on creating watches, see the *Watches User Guide* (5189).

### Specifying Device Type Names for Device Identification

The DeviceType attribute (0x23000e) is a text string displayed below the device icon that identifies the type of device being modeled. SPECTRUM provides the ability to search, filter, and report on device models using the DeviceType attribute, which gives you a finer level of granularity when managing your device network infrastructure.

The Device Type Identification (DTI) application allows you to maintain a user-defined list of system object identifiers (system object IDs) and corresponding device type names. When you create or modify one of these entries, the DeviceType attribute for all device models with the given system object ID is set to the corresponding device type name. This is the case for both existing and future device models. Coupled with the GnSNMPDev model type's flexibility, this feature makes it possible to model and monitor any SNMP-compliant device in the network even if a specific SPECTRUM management module is not available.

The Device Type Identification list also contains unregistered devices, that is, devices that have been modeled using Discovery or Model by IP that have system object IDs in the Device Type Identification list but not corresponding device type names.
**Tip:** For quick identification, unregistered devices are displayed in the list in bold.

You can use the unregistered devices in the Device Type Identification list to simplify the initial task of setting up entries for all devices modeled with GnSNMPDev. In other words, instead of trying to determine all of the devices that use the GnSNMPDev model type, first model the devices. Once the devices are modeled, their system object IDs are added to the Device Type Identification list, and the entries are displayed in bold. You can then filter and sort the list as needed and specify device type names for the unregistered devices modeled with GnSNMPDev.

**Note:** The mappings in the Device Type Identification list are preserved regardless of Service Pack upgrades or database migrations.

To start the Device Type Application, in OneClick, click Tools > Utilities > Device Type Identification. For complete information on using the application, including information on using the application in a fault-tolerant environment or a distributed environment, see the *Modeling Your IT Infrastructure Administrator Guide (5167)*.
Chapter 4: Developing a New Management Module

If you determine that the customization options outlined in the preceding chapter do not meet your device management needs, consider creating a new device model type or a new application model type using GnSNMPDev.

Choosing to Develop a New Management Module

A device model type and its associated interface and application model types collectively model a device. When you consider creating a new device model type or simply adding to the functionality of the GnSNMPDev device model type, you must understand the functional components of the device in order to effectively expand SPECTRUM’s management of the device.

There are many device functions that are supported by both the GnSNMPDev device model type and the interface and application models known to the GnSNMPDev model type. These include functionality from both proprietary and standard MIBs. Before creating a new device or application model type, it is best to identify the functionality of the device that is already supported by GnSNMPDev. For example, if a device’s interfaces map one-to-one with physical ports on a single board, GnSNMPDev can support this device, without enhancement, via its built-in support for MIB-II interfaces in the Snmp2_Agent application model.

To find out how GnSNMPDev will support the device, model the device in OneClick using its IP address. SPECTRUM automatically finds the model type most appropriate for the device. If there is not a specific model type, SPECTRUM will choose the GnSNMPDev model type and instantiate a GnSNMPDev model to represent the device.

Once you have established the type of support that SPECTRUM can provide for your device by default and considered the customizations described in Chapter 3: Customizing the GnSNMPDev Management Module, you can decide if further customization is necessary in order to manage your device properly. The following sections outline some scenarios in which expanded support might be necessary.
Additional MIB Support

If access to additional MIBs is required for your device management needs, there are a number of ways in which MIB objects can be made available to a device model:

- The first and recommended method is to import the new MIB directly into the SpectroSERVER using the import mechanism provided in MIB Tools. For more information on this method, see Adding Support for Additional MIBs on page 7. For complete instructions on MIB Tools, see the Modeling Your IT Infrastructure Administrator Guide (5167).
- The second method is to create a new device model type to represent your device and derive the necessary MIBs into the device model type. See Creating a New Device Model Type on page 14 for complete instructions.
- The third method is to create a new application model type that provides access to the new MIB. See Creating a New Application Model Type on page 25 for complete instructions.

Unique Trap Mapping

You will need to create a new device model type if the device you are modeling requires that you implement unique trap processing in response to a common trap.

For example, assume that you want your network’s core routers to generate a major alarm in response to an authentication failure, but you want all other devices to generate a minor alarm in response to the same. By default, SPECTRUM generates a minor alarm in response to an authenticationFailure trap (see Traps, Events, and Alarms on page 6), so to meet your requirements, you must create a new device model type and create support for the trap in the device model type’s event and alert configuration files. This support would override SPECTRUM’s default processing for the authenticationFailure trap for this model type only.

See Creating a New Device Model Type on page 14 for instructions on creating a new device model type. See Adding Support for Additional Traps on page 36 for information on trap processing.
Unique Watches

You can generate events and alarms based on the results of a watch. The GnSNMPDev model type provides a number of predefined watches that can be turned on and off on a per model basis.

If you need to customize the watch implementation on the models that represent your device, you can do this for each applicable GnSNMPDev model that is instantiated. However, to avoid having to repeat this customization on each model, you can also create your own device model type to implement the customized watch configuration.

All of the information that makes up a watch is stored as attributes in the model type specified in the watch. The only exception to this is the probable cause information created for an alarm resulting from the watch. This information is stored in a model type called ProbCause.

Because you have not created the ProbCause model type with your Developer ID, you do not have permissions to export and distribute it with the other components of your management module. This means that the probable cause information that relates to the watches you have created will not be distributed. To solve this problem, you must derive a new model type from the ProbCause model type. The probable cause information for any watches created for any of your management modules will automatically be stored in this derived model type. Because you have created this derived model type, you can distribute it with your management module.

For information on creating a device model type, see Creating a New Device Model Type on page 14. For information on creating watches, see the Watches User Guide (5189).

Interface Model Creation

If your device does not advertise its interface (port) information in the MIB-II standard interface table, but instead uses information from a proprietary MIB, SPECTRUM will not be able to model the interfaces on your device.

Without interface models, you will be unable to resolve connections to the interface level, and you will be unable to monitor the status of each interface. To work around this, you must create a new application model type that includes support for the proprietary MIB with the interface information. See Creating a New Application Model Type on page 25.
Creating a New Device Model Type

This section describes the factors to consider and steps to perform when creating a new device model type. These include the following:

- Understanding the database derivation and MIB requirements
- Creating the model type and setting required attributes
- Selecting discovery and identification mechanisms
- Making desired customizations
- Making the new model type distributable to other SPECTRUM hosts

Designing a New Device Model Type

The device model type database architecture used for developing new device model types is an organized structure where all proprietary MIBs are imported into separate MIB model types and derived directly into the device model type as shown in the following figure.
This scheme has the following advantages:

- It allows a single MIB to be derived into multiple model types (for example, multiple device model types or a device and an application model type) while maintaining the attribute IDs.
- It allows vendor MIBs to be organized in a single collection.
- It allows for convenient access to MIB information directly from the device model. For example, OneClick views, watches, and logging and polling information about proprietary vendor attributes can all be accessed from the device model.

If there are additional MIBs that your new device model needs to access from SPECTRUM, the simplest method to use is the MIB import mechanism available in MIB Tools. This mechanism creates attributes from these objects and makes them available to all models that represent a device in the SpectroSERVER’s database. See the *Modeling Your IT Infrastructure Administrator Guide (5167)* for more information and instructions on using this functionality. Note that the MIB import mechanism will distribute the new MIB across all SpectroSERVERS in a distributed environment.

**Creating a New Device Model Type**

Once you have determined the desired database scheme, use the Model Type Editor to create your model types. All device model types should be derived from the GnSNMPDev model type. For instructions on using the Model Type Editor, see the *Model Type Editor User Guide (0659)*.

**Note:** When you are using the Model Type Editor to create a new device model type, it is important to remember to correctly set the model type flags for the new model type. For more information, see *Setting Model Type Flags on page 16*.

**Configuring a New Device Model Type**

There are a few steps involved in configuring a new device model type. They include the following:

- Setting model type flags
- Setting attribute values
- Mapping a device or a device family to the new device model type
- Configuring serial number handling

**Note:** The following sections provide high-level information on these configuration steps. For step-by-step information on using the Model Type Editor to complete each step, see the *Model Type Editor User Guide (0659)*.
Setting Model Type Flags

It is important to set the value of various model type flags to ensure that models of this model type behave properly within SPECTRUM. Each flag represents a Boolean value and can either be selected (set to TRUE) or deselected (set to FALSE).

In most cases you should select the Visible, Instantiable, and Derivable flags.

- If the Visible flag is set to TRUE, the model type is visible to all Model Type Editor users. If the Visible flag is set to FALSE, the model type is only visible to a user with the same developer ID as the one used to create the model type.
- If the Instantiable flag is set to TRUE, you can instantiate a model of this model type in OneClick.
- If the Derivable flag is set to TRUE, this model type can be used as a base for other model types.

In most cases you should deselect (set to FALSE) the No Destroy, Unique, and Required flags.

- If the No Destroy flag is set to TRUE, users are not able to destroy a model of this type in OneClick.
- If the Unique flag is set to TRUE, only one model of this model type can be instantiated in OneClick.
- If the Required flag is set to TRUE, then a model of this model type must always exist in the SpectroSERVER database.

Setting Attribute Values

Once you have created your new device model type, you will need to use the Model Type Editor to set the default value of several attributes. Some of these settings are used to configure various built-in capabilities inherited by deriving from the GnSNMPDev model type. The following table describes the attributes and settings.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompanyName</td>
<td>The name of the company developing the management module.</td>
</tr>
<tr>
<td>Description</td>
<td>There are two description attributes, one in the MMDriver group and one in the CommonInfo group. The Description attribute in the MMDriver group has a default value of Generic SNMP Device Management Module. It is recommended that you reset this default value with a basic description of your management module. The Description attribute in the CommonInfo group can be filled in with the identical text or left empty.</td>
</tr>
<tr>
<td><strong>Desc_Key_Word</strong></td>
<td>In the case that the <code>System_Desc_Verify</code> or <code>Vendor_Object_ID</code> discovery mechanisms identify multiple device model types, <code>sysDescr</code> can be searched for a substring match from the value of this attribute. For more information, see Mapping Devices to the New Device Model Type Using <code>sysObjectID</code> and Strings in <code>sysDescr</code> on page 21.</td>
</tr>
<tr>
<td><strong>DeviceNameList</strong></td>
<td>Used in conjunction with <code>SysOIDVerifyList</code>. Populate this list attribute with text strings identifying the devices that will be modeled with this model type. These will be matched up against the entries found in <code>SysOIDVerifyList</code>. Device discovery will run through the list of <code>sysObjectIDs</code> and if a match is found, the coinciding text string entry will be populated into the <code>DeviceType</code> attribute. For more information, see Mapping Devices to the New Device Model Type Using Unique Values in <code>sysObjectID</code> on page 20.</td>
</tr>
<tr>
<td><strong>DeviceSerialAttr</strong></td>
<td>Set this to the attribute ID of the external attribute which contains the serial number of the device. When the model is created, it will read this external attribute and write it into <code>Serial_Number</code>.</td>
</tr>
<tr>
<td><strong>DeviceType</strong></td>
<td>A description that identifies the device. A default value is required for this attribute when the <code>DeviceNameList</code> identification mechanism is not used. By setting the default value, this will guarantee a value will be present for displaying, sorting and filtering.</td>
</tr>
</tbody>
</table>
| **DeviceTypeDiscEnable** | Used to enable or disable `DeviceType` naming intelligence. The default value of true is appropriate for most device model types. However, it may be desirable to set this value to false under either of the following conditions:  
- A new device model type has been derived from a base model type other than GnSNMPDev that has specialized `DeviceType` naming intelligence that is inappropriate for the derived device model type  
- A more appropriate `DeviceType` name can be set in the catalog for the derived model type |
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disposable_Precedence</td>
<td>This attribute is evaluated at device model type discovery time when more than one model type is identified as a possible candidate. The higher value will be the chosen model type. This value is also evaluated when a model is created that has the same MAC address as a previously existing model. In this case, both models’ disposable_precedence attributes are evaluated. The model with the higher value will replace the existing model by stealing its CONNECTs associations.</td>
</tr>
<tr>
<td>Enable_IH_Enterprise_Disc</td>
<td>Used to enable or disable the automated setting of the Manufacturer and App_Manufacturer attributes based on the enterprise ID term of the device’s sysObjectID. The default value is true, which is appropriate for GnSNMPDev as it is used to model devices from various manufacturers. However, for a new device model type derived from GnSNMPDev where the manufacturer of the modeled device is known, CA recommends that you set the value of Enable_IH_Enterprise_Disc to false, and set the default values of the Manufacturer and App_Manufacturer attributes to the appropriate names.</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Name of the vendor that manufactures the device.</td>
</tr>
<tr>
<td>MMName</td>
<td>The name of the management module.</td>
</tr>
<tr>
<td>MMPartNumber</td>
<td>The part number you will be giving to the management module.</td>
</tr>
<tr>
<td>System_Desc_Verify</td>
<td>This provides a device model type discovery mechanism by which the sysDescr is parsed for firmware version information. Clear this default value if you are not using this discovery mechanism, as it cause problems for the other discovery methods if it set. For more information, see Mapping Devices to the New Device Model Type Using Firmware Version Strings in sysDesc on page 21.</td>
</tr>
<tr>
<td>System_Oid_Verify</td>
<td>This is a legacy attribute. Please refer to SysOIDVerifyList.</td>
</tr>
</tbody>
</table>
| **SysOIDVerifyList** | Used in conjunction with DeviceNameList. Populating this list attribute with sysObjectID values allows device model type discovery intelligence to match the list against the device’s sysObjectID value. If a match occurs, this model type will be chosen as a possible contender to be used for modeling.

For more information, see [Mapping Devices to the New Device Model Type Using Unique Values in sysObjectID on page 20](#).
|
| **Vendor_Name** | The name of the company developing the management module.
|
| **Vendor_Object_ID** | This provides a device model type discovery mechanism by which a partial sysObjectID match identifies a device model type.

For more information, see [Mapping Devices to the New Device Model Type Using sysObjectID and Strings in sysDesc on page 21](#).
|
| **VendorIDVerifyList** | Used during discovery in conjunction with VendorOIDVerifyList to map devices to device model types based on whether the device supports specific MIB objects.

Specify the list of enterprise IDs to compare against the device to model; if a match is found, the corresponding MIB object specified in VendorOIDVerifyList is read from the device.

For more information, see [Mapping Devices to the New Device Model Type Using a MIB Object on page 22](#).
|
| **VendorOIDVerifyList** | Used during discovery in conjunction with VendorIDVerifyList to map devices to device model types based on whether the device supports specific MIB objects.

Specify the list of attribute IDs for the MIB objects to read from the device.

For more information, see [Mapping Devices to the New Device Model Type Using a MIB Object on page 22](#).
|
| **Verify_Mismatch_Model** | Set to TRUE. This attribute will cause SPECTRUM to perform checks for a device model type match with the device being modeled. |
Mapping a Device or a Device Family to the New Device Model Type

It is important to uniquely identify a device on the network. Most commonly this is done through the MIB-II object `sysObjectID`. Most vendors will assign a unique `sysObjectID` value for a particular device, thus creating a one-to-one mapping. Many vendors advertise this information in a Products MIB, which is an excellent source to obtain the mapping of `sysObjectID` to device model type.

You can uniquely identify a device in several ways using the Model Type Editor:

- If your device provides a unique `sysObjectID` value, then use the process described in Mapping Devices to the New Device Model Type Using Unique Values in `sysObjectID` on page 20.
- If your device does not provide a unique `sysObjectID` but does provide a unique substring within `sysDescr`, then use the process described in Mapping Devices to the New Device Model Type Using `sysObjectID` and Strings in `sysDescr` on page 21.
- If your device does not provide a unique `sysObjectID` but does provide a firmware version text string in `sysDescr`, then use the process described in Mapping Devices to the New Device Model Type Using Firmware Version Strings in `sysDescr` on page 21.
- If your device does not provide any of the previously described information, you can map the device to a device model type based on whether the device supports specific MIB objects in a proprietary MIB. Refer to Mapping Devices to the New Device Model Type Using a MIB Object on page 22.

Mapping Devices to the New Device Model Type Using Unique Values in `sysObjectID`

If your device has a unique `sysObjectID` value, you must relate your new device model type to the `sysObjectID` to ensure that SPECTRUM selects the new device model type to represent the device. To do this, add the `sysObjectID` value to the `SysOIDVerifyList` model type attribute. If the new device model type represents a family of devices, then add each `sysObjectID` value.

**Note:** If another model type contains the same `sysObjectID` value in its `SysOIDVerifyList` attribute, it is possible that SPECTRUM will choose the other model type to represent a device with this `sysObjectID`. If this occurs, you should change the `disposable_precedence` attribute value on your device model type to a higher value than that of the other model type. For example, if the other model type has a `disposable_precedence` value of 10, change the `disposable_precedence` value on your model type to 11.

In order to provide identification to your model, you can configure the model type to display a different device name for each of the devices that the model type is designed to support. For example, assume your device model type represents the 8480 series of switches made by MySwitch, Inc. Instead of seeing the device name `MySwitch_8480XX` for all of the switches in the 8480 family, you want to display the model number of the switch, as appropriate. If SPECTRUM is modeling an 8480-09 switch, the model should display the device name `MySwitch_8480-09`. If SPECTRUM is modeling an 06 switch, the model should display the device name `MySwitch_8480-06`.
To set up the relevant attributes to enable this functionality on the model type:

1. Set the `SysOIDVerifyList` attribute equal to the `sysObjectID(s)` of the devices that the model type will represent.
2. Set the `DeviceNameList` attribute equal to the device names that apply to each `sysObjectID` listed in the `SysOIDVerifyList` attribute.
   Specify the same number of names in the `DeviceNameList` attribute as there are `sysObjectIDs` listed in the `SysOIDVerifyList` attribute. The names should be listed in the same order as their corresponding `sysObjectIDs`.
3. Clear the `System_Desc_Verify` default value.
   **Note:** The `DeviceNameList` attribute will only work for device model types that use the `SysOIDVerifyList` attribute model type discovery mechanism. Make sure that both lists have the same number of entries. Otherwise, the `DeviceType` attribute will not be set correctly.

**Mapping Devices to the New Device Model Type Using sysObjectID and Strings in sysDesc**

If your device does not provide a unique `sysObjectID`, a partial or complete match of the `sysObjectID` in combination with a `sysDescr` substring may provide unique identification.

To set up the relevant attributes to enable this functionality on the model type:

1. Set the `Vendor_Object_ID` attribute equal to the partial or complete `sysObjectID` value returned by your device.
   **Note:** Only the first 7 terms (up to the enterprise ID) will be used for comparison.
2. Set the `Desc_Key_Word` attribute equal to the unique, partial `sysDescr` value returned by your device.
3. Set the `DeviceType` attribute equal to the desired identification string.

**Mapping Devices to the New Device Model Type Using Firmware Version Strings in sysDesc**

If your device does not provide a unique `sysObjectID` or a unique sub-string within `sysDescr`, check if `sysDescr` provides a unique firmware version. This discovery mechanism searches the `sysDescr` value for either “Version” or “Rev.” If one of these strings is found, the value of `System_Desc_Verify` is compared against the text that follows these key words. If a match is found, the device model type will be chosen. In the case where multiple model types have the same `System_Desc_Verify` value, a substring in `sysDescr` can be compared by setting the `Desc_Key_Word`.
Creating a New Device Model Type

To set up the relevant attributes to enable this functionality on the model type:

1. Set the `System_Desc_Verify` attribute equal to the contents of `sysDescr` that follow the key text noted above.
2. Set the `Desc_Key_Word` attribute equal to the unique, partial `sysDescr` value returned by your device.
3. Set the `DeviceType` attribute equal to the desired identification string.

Mapping Devices to the New Device Model Type Using a MIB Object

If your device does not provide a unique `sysObjectID` or a unique sub-string or firmware version within `sysDescr`, check if it supports a proprietary MIB. You can map the device to a device model type based on whether the device supports specific MIB objects.

This discovery mechanism compares the enterprise ID of the device against each enterprise ID specified in the `VendorIDVerifyList` attribute. If a match is found, the MIB object specified in the same instance of the `VendorOIDVerifyList` attribute is read from the device. If the SNMP read succeeds, this model type is added to the list of model type candidates (from which the model type with the highest `disposable_precedence` attribute value is ultimately selected). The enterprise ID match mechanism is done for performance reasons so that the SNMP read is only initiated for targeted devices.

To set up the relevant attributes to enable this functionality on the model type:

1. Add the enterprise ID of the device to model with this model type to the `VendorIDVerifyList` attribute.
2. Add the attribute ID of the MIB object to read from the device as the corresponding instance in the `VendorOIDVerifyList` attribute.
3. Repeat the preceding steps for each enterprise ID/attribute ID pair to evaluate in conjunction with one another.
4. Set the `DeviceType` attribute equal to the desired identification string.

Discovery and Identification Flowchart

The flowchart at the end of this section identifies the steps that SPECTRUM takes to determine the device model type to represent a device. The flowchart includes the most common discovery and identification mechanisms:

- Using unique values in `sysObjectID`
- Using strings in `sysObjectID` and `sysDesc`
- Using firmware version strings in `sysDesc`
Note: You can also map a device (or a device family) to a new device model type based on whether the device supports specific MIB objects in a proprietary MIB. For more information, see Mapping Devices to the New Device Model Type Using a MIB Object on page 22.
Creating a New Device Model Type

SPECTRUM queries the device and obtains the values for the device’s sysObjectID and sysDescr attributes. SPECTRUM looks in the modeling catalog to determine the model types whose System_Oid_Verify attribute value or SysOIDVerifyList list attribute values match the sysObjectID obtained from the device.

Is a match found?

Yes

Use this model type.

No

SPECTRUM parses the device’s sysDescr value for ‘Revi’ and ‘Version,’ then matches the text following to device model types System_Desc_Verify attribute value.

SPECTRUM searches for model types whose Vendor_Object_ID attribute value matches a subset of the sysObjectID obtained from the device.

SPECTRUM looks at the Desc_Key_Word attribute of the model types in the catalog. If this key word occurs anywhere in the sysDescr of the device, this model is considered a match.

Use this model type.

How many matches are found?

1

Use this model type.

0

How many matches are found?

SPECTRUM looks at the disposable_precedence attribute value for each model type.

How many matches are found?

1

Use this model type.

0

How many matches are found?

SPECTRUM will call various management module specific methods which will check for a match dependent upon that management module’s particular functionality.

Use this model type.

How many matches are found?

1

Use this model type.

0

Does one model have the highest value?

Yes

Use this model type.

No

Use the first model type found.

Use the GnsSNMPDev model type.
Configuring Serial Number Handling

Device model types contain an attribute used for setting and displaying the serial number of the modeled device: Serial Number (0x10030). You can enter the appropriate serial number value in any of the views where the attribute is displayed, or if the serial number is available as an external attribute from the device, you can configure the model type to automatically retrieve this value and set the Serial_Number attribute. To do this:

1. Ensure that the external attribute that contains the serial number is not a list attribute and is of type TEXT_STRING or OCTET_STRING.
2. Use the Model Type Editor to set the value of the device model type’s DeviceSerialAttr (0x3d0063) attribute equal to the ID of the external attribute that contains the serial number.

When a model of this model type is instantiated, SPECTRUM will set the Serial_Number attribute equal to the value of the external attribute containing the serial number.

Creating a New Application Model Type

This section describes how to expand support for a device using application model types. All application model types are derived from a series of standard model types called derivation points. An Application will often correspond to the functionality of a MIB.

Understanding Derivation Points and Model Fragments

Derivation points must be selected and used as base model types for new application model types. All of these derivation points have functionality designed to support different types of applications. When you derive a new model type from one or more of these derivation points, the model type will inherit the functionality of those derivation points.

Some derivation points require the use of model fragments. The model fragments that are available are model types that have specific inference handlers attached to them. These inference handlers provide the model fragments with certain behaviors and intelligence, such as the ability to create port or board models. In order to use the functionality provided by these inference handlers, you must map attribute IDs from the model type representing the MIB to specific model fragment attribute values.

Usually, model fragments are included as base model types for the GnSNMPDev derivation points that require them. However, you may find it necessary to add a model fragment as a base model type to your new model type to take advantage of the capabilities of the inference handler attached to the model fragment.
The following model types can be used as application derivation points:
- GnSNMPMibDerPt
- GnSNMPAppDerPt
- GnChassisDerPt
- GnDevIODerPt
- GnRelayDerPt

The following figure shows the application derivation point hierarchy and sample derived model types. The lines connecting the model types denote the inheritance structure. Note that you will want to select only one of the dotted line paths for your new application model type derivation hierarchy.

The derivation points for application model types are all designed to provide you with specific functionality. GnChassisDerPt, GnDevIODerPt, and GnRelayDerPt each have model fragments that enhance this functionality. The following table shows each derivation point, the application model type it is used to create, and its associated model fragments.

<table>
<thead>
<tr>
<th>Derivation Point</th>
<th>Model Type</th>
<th>Associated Model Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GnSNMPMibDerPt</td>
<td>MIB Model Type</td>
<td>N/A</td>
</tr>
<tr>
<td>GnSNMPAppDerPt</td>
<td>Application model type that does not need to manage ports or boards.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Choosing a Derivation Point

`GnSNMPAppDerPt` includes the basic functionality needed for an application model type. `GnChassisDerPt`, `GnDevIODerPt`, and `GnRelayDerPt` are derived from `GnSNMPAppDerPt` and, therefore, inherit this functionality. Each also includes some specialized functionality geared towards managing ports and boards.

If your device does not need to manage ports and boards and you are only interested in expanding support, use `GnSNMPAppDerPt` to derive your application model type. For complete instructions on how to do this, see Creating Application Model Types on page 30.

If your device uses other MIBs to extend the functionality of MIB-II in order to manage ports and boards, you will need to use `GnChassisDerPt`, `GnDevIODerPt`, or `GnRelayDerPt`. Each of these derivation points uses model fragments that contain the attributes and intelligence needed to create port models. The following section, Board and Port Considerations on page 27, describes how to decide which derivation point is most appropriate for your port or board model type.

Board and Port Considerations

As a general rule, if the device you are attempting to model is a chassis (a device that has multiple modules or boards that can be inserted and removed), then you would use the `GnChassisDerPt` and `GnRelayDerPt` derivation points to create the application model type. These two derivation points will be used to model both the boards and ports found in the device. The intelligence of these derivation points will create both board and port models.

If the device you are modeling is not a chassis, then you would build your application model type from the `GnDevIODerPt` derivation point. The intelligence of this derivation point will create only port models (no boards) and associate the port models with the device model.
The structure and content of the relevant MIBs is important to consider. Chassis and data relay MIBs generally have a standard structure. A chassis MIB usually has a slot and board table. The index of the table represents which slot of the chassis the board is plugged into. A data relay MIB usually has two tables: a board table and a port table. The board table is indexed by which slot the board is plugged into. The port table typically has two indexes: a board index and a port number. Additionally, vendors have devised several variations to the standard structures described above.

Port-Oriented Devices

You will generally use the GnDevIODerPt to model port-oriented, non-chassis devices. You will find that most MIBs for these port-oriented devices conform to the structural requirements necessary to use GnDevIODerPt. The MIB must contain a port table, with at least one index, the port number. The intelligence associated with the GnDevIODerPt will execute a read_next (a read_next is analogous to the get_next SNMP call) on this attribute. For each successful read of the index attribute, a port model with the appropriate instance ID will be instantiated.

Chassis Devices

The structure of the MIBs associated with chassis devices is much more varied. The best way to examine the variations and how they affect the modeling of the device is to view what is required by the intelligence associated with the GnChassisDerPt and the GnRelayDerPt derivation points.

GnChassisDerPt

The GnChassisDerPt is used to create an application model type that will become the chassis manager application. This application will be responsible for the creation and management of board models in the SpectroSERVER database. This chassis manager relies on three attributes (usually list attributes) for the information it needs:

- slot index
- board type
- board label

There can only be one chassis manager application instantiated or managed by the main device model. The chassis manager intelligence is expecting the MIB to have a slot or board table indexed by an integer value representing the slot into which a particular board is plugged. The intelligence will perform a read_next on this slot_index attribute. For each successful read, the intelligence will create a model in the database to represent that board. Because the intelligence can only reference one index value, all boards in the chassis must have an entry in this single table of the chassis MIB.
In addition to finding which slot a board is plugged into, the manager intelligence will need to determine the board’s type and label the board correctly. This information is determined by the board type and board label attributes. It is not necessary that these attributes exist in the same table as the slot index attribute. All that is required is that the attributes exist in a table with the same indexing scheme as the table used to discover the boards.

It is possible that the MIB will have all the board information in non-list attributes rather than in a table. In this case, the information supplied within the MIB is for a single board, and the slot index value is not really an index into a table, but simply an integer attribute that will return the slot that the board is located in. The chassis manager intelligence will test the slot index attribute. If it is a non-list attribute, a read will be used instead of a read_next to get the board number. If the slot index attribute is not a list attribute, the board type and board label attributes will not be list attributes.

**GnRelayDerPt**

The intelligence of GnRelayDerPt is used to model the ports on a chassis. This derivation point can be used in combination with GnChassisDerPt to create one application model, or it can be used on its own to create an application model separate from the chassis manager.

The term **chassis support application** is used to describe an application built with GnRelayDerPt. This is because it provides support to the chassis manager application (such as modeling the ports for each board). Unlike the chassis manager application, you can have multiple chassis support applications instantiated under the main device model. This becomes important when you consider a chassis that has boards supporting different protocols.

Although all the boards may show up in the chassis’ slot table, the data relay component of each board may be managed by a MIB corresponding to the appropriate protocol. It is necessary to have each of these protocol-dependent MIBs modeled as separate application models (built from the GnRelayDerPt derivation point) so that the ports found on each of the boards can be discovered and modeled.

The typical structure of a data relay MIB has two tables: a board table and a port table. The board table should not to be confused with the slot table used with the chassis manager, although in some cases they can be the same tables. The board table found in the data relay MIB will have an entry for each board supported by the MIB, typically indexed by the position of the board in the chassis. For example, if the data relay MIB in question is an Ethernet MIB, then any board that supports the Ethernet protocol (typically a repeater board) will have an entry in this MIB’s board table. If a FDDI board is plugged into the chassis, the board will create an entry in the common slot table, but this new board will not show up in the Ethernet MIB’s board table. Instead, it will show up in the board table of the FDDI MIB.
Along with the board table, the data relay MIB will have a port table. For each port supported by the MIB, there will be an entry in this table. The tables often contain the status and statistical information for each port. The port table contains two indices: a board index and a port index. Because the port table contains a board index, the chassis support intelligence can associate the port models with the appropriate board models; the board index supplies the mapping of a port to a board.

_GnDataRelay_MF_ is the model fragment within the _GnRelayDerPt_ derivation point that contains the attributes and intelligence that are used to model each board’s ports and associate those port models with the appropriate board model. The intelligence associated with the _GnDataRelay_MF_ model fragment works with only one board table and one port table. In the majority of cases, this is not a problem because this is the typical structure of a data relay MIB. If your data relay MIB contains sets of tables—for example, a set of board and port tables for each of the major protocols—then you must separate these tables or groups of the MIB into separate model types. To do this, use each model type as a base to the appropriate application built with the _GnRelayDerPt_.

There may be some cases where the data relay MIB does not have the typical structure of both a board table and a port table, with the port table indexed to provide the physical mapping of ports to boards. This can be the case when the chassis device uses a MIB with a different indexing scheme for accessing the port information. An example of this would be the FDDI MIB that indexes the port table by the _SMTIndex_ and the _PortIndex_. The _SMTIndex_ has nothing to do with which board the FDDI port is physically located.

This situation can also be created if a vendor reuses a MIB from another device that it manufactures. The original device that the MIB was designed to manage was a port-oriented device (no boards, just ports). The vendor supplies the same functionality in a board that can be plugged into its chassis, and has decided to use the original MIB to manage the ports on that board. Its port table does not contain a board index so there is no means of determining which board(s) has which port.

In this case, you should implement the _DataRelay_MF_ model fragment functionality as you would with a port-oriented device.

**Creating Application Model Types**

The following is the core set of tasks that you must accomplish when creating an application model type:

1. Import the required MIBs.
2. Derive the application model type.
3. Set up application model discovery.
4. Set the model name.
5. Map the model fragments.
6. Set the model type flags.
You use the Model Type Editor to accomplish each of these tasks. The following sections are designed to help you understand why the above tasks are necessary. For complete information on using the Model Type Editor, see the Model Type Editor User Guide (0659).

Importing Required MIBs

When you create an application model type, you may find that the MIB model type already exists in SPECTRUM, or you might need to provide access to the new MIB. If you need to provide access to the new MIB, you can do it in one of two ways:

- Use the Model Type Editor to import the MIB directly into the new application model type
- Create a MIB model type

If the MIB will be derived into multiple model types, it may be advantageous to derive the MIB into a separate model type that can be used as a derivation point, thus allowing the attribute IDs to be maintained across the model types. Organizing this MIB model type under a new or existing vendor model type helps to keep the database organized, as described in Designing a New Device Model Type on page 14.

To create a MIB model type, derive a new model type from GnSNMPMibDerPt (see Designing a New Device Model Type on page 14). Import the compiled MIB, and enter the proper SMI (Structure of Management Information) path.

Note: If the wrong SMI Path is used, the Model Type Editor will not produce an error. However, when viewing imported attributes, the OID Prefix value will be incorrect.

Deriving the Application Model Type

To derive an application model type, in the Model Type Editor, set the GnSNMPAppDerPt model type as the current model type, and then create a new derived model type.

After you have created the new application model type, if you have created a MIB model type, you should now add it as a base model type to the new application model type.

The new application model type should now contain two base model types:

- GnSNMPAppDerPt model type
- custom MIB model type
Creating a New Application Model Type

Setting Up Application Model Discovery

When a device model for a specific device is instantiated, SPECTRUM queries the Model Type catalog. Most application model types that are derived from GnSNMPAppDerPt are queried, and the value of each of these model types’ default_attr or default_attr_list attribute is retrieved. SPECTRUM then queries those attributes on the device’s MIB. When a match is found between an attribute value retrieved from the application model type and the corresponding attribute value retrieved from the MIB, SPECTRUM instantiates a model of this model type.

You can use either the default_attr_list or default_attr to specify attribute IDs from attributes of a MIB model type. The default_attr_list attribute allows you to specify multiple attribute IDs, and the default_attr attribute allows you to specify one attribute ID. Each attribute allows SPECTRUM to identify the application model type that represents the MIB’s functionality.

SPECTRUM performs a query of the attributes whose attribute ID is contained in the default_attr or default_attr_list. If default_attr_list is used, SPECTRUM will go through the list of attribute IDs and use the first supported attribute ID found to instantiate that application model to represent the MIB’s functionality.

The default_attr_list attribute can be helpful if you have a device that supports just one table in a MIB rather than the entire MIB, and another device that supports other objects in the same MIB, but not in the particular table that the other device supports. In this scenario, using the default_attr_list attribute to specify multiple attribute IDs ensures that the application model type representing the MIB will be instantiated for both devices even though they do not support the same objects in the MIB.

Note: You must set the default_attr or default_attr_list in all application model types. When choosing a value, it is recommended that you use an attribute from the MIB model type that represents a mandatory, non-list, external MIB variable. This is especially important when creating a chassis application.

To specify a value for default_attr

1. Find the MIB attribute for the application model type with which you are working.
2. Use the attribute ID of this attribute to set the value of the default_attr attribute in the application model type.

You will need to look specifically at the attributes of the model type that represents the MIB. You can find the attribute IDs of a model type’s attributes on the Attributes tab in the Model Type Editor.
Creating a New Application Model Type

To specify values for default_attr_list

1. Find the MIB attributes for the application model type with which you are working.
2. Use the attribute IDs of these attributes to specify values in the default_attr_list attribute in the application model type.
   You can find the attribute IDs of a model type’s attributes on the Attributes tab in the Model Type Editor.
3. Set the Model_Group attribute to the decimal value of the application model’s model type handle.
   **Note:** It is important to make sure that the value of Model_Group is set appropriately. If Model_Group is set to 0, SPECTRUM will only use the default_attr attribute to identify the application model type that represents the MIB’s functionality.

Setting the Model Name

Set the application model type’s Model_Name attribute to the appropriate value. By default, this value will be used as the model name for any application model of this type.

Mapping Model Fragments

If your new application model type is derived from GnChassisDerPt, GnDevIODerPt, or GnRelayDerPt, you must work with the model fragments that correspond to these model types to ensure the correct operation of port and board management. For a model fragment to function properly, you must map MIB attribute values from the application model type to model fragment attribute values using the Model Type Editor. This gives the model fragment access to information from the MIB that it uses to create and manage ports, boards, and interfaces.

For example, one of the attributes required for the GnChassis_MF model fragment used with the GnChassisDerPt derivation point is the boardIndex_Attr. This attribute allows the model fragment to discover which boards are present in a chassis. The boardIndex_Attr needs to be set to the index attribute value in the board (group) table of the chassis or repeater MIB. The index attribute usually returns an integer value or a series of values that represents a board number.

Certain derivation points have associated model fragments. The attributes associated with that model fragment are available to any model type based on these derivation points. If you need the functionality of a model fragment that is not included with one of your base model types, include that model fragment as a base model type.
Creating a New Application Model Type

Setting Model Type Flags

When creating an application model type, it is important to set the value of a few different flags to ensure that models of this model type behave properly within SPECTRUM. These flags are available on the Flags tab of the current model type in the Model Type Editor. Each flag represents a Boolean value and can either be selected (set to \texttt{TRUE}) or deselected (set to \texttt{FALSE}).

In most cases, you should select the \texttt{Visible}, \texttt{Instantiable}, and \texttt{Derivable} flags.

- If the \texttt{Visible} flag is set to \texttt{TRUE}, the model type is visible to all Model Type Editor users. If the \texttt{Visible} flag is set to \texttt{FALSE}, the model type is only visible to a user with the same developer ID as the one used to create the model type.
- If the \texttt{Instantiable} flag is set to \texttt{TRUE}, you can instantiate a model of this model type in OneClick.
- If the \texttt{Derivable} flag is set to \texttt{TRUE}, this model type can be used as a base for other model types.

In most cases, you should deselect (set to \texttt{FALSE}) the \texttt{No Destroy}, \texttt{Unique}, and \texttt{Required} flags.

- If the \texttt{No Destroy} flag is set to \texttt{TRUE}, users are not able to destroy a model of this type in OneClick.
- If the \texttt{Unique} flag is set to \texttt{TRUE}, only one model of this model type can be instantiated in OneClick.
- If the \texttt{Required} flag is set to \texttt{TRUE}, then a model of this model type must always exist in the SpectroSERVER database.

Modeling Ports and Boards

When you create application model types from \texttt{GnChassisDerPt}, \texttt{GnDevIODerPt}, and \texttt{GnRelayDerPt}, these applications automatically create the necessary port and board models needed to represent your device. SPECTRUM generally uses two model types to model these boards and ports: \texttt{GnModule} and \texttt{GnPort}. It is possible to derive new model types from these model types for customization purposes.

In OneClick, you can view the ports for a device on the Interfaces tab in the Component Details pane. To view the boards for a device, use the Locater tab to search for the boards by model type name.

Modeling Boards with \texttt{GnModule}

Typically a board is modeled for one reason, namely, to be a container for the port models that are physically located on that board. In GnSNMPDev’s chassis support, the \texttt{GnModule} model type is used to model many different types of boards.
All new board model types must be derived from the GnModule model type. GnModule has two attributes that help to define what type of board is being represented by a particular model:

- \textit{gnType}: This attribute provides the board type as read from the chassis slot table. When each GnModule model type is instantiated, the chassis manager intelligence fills in the gnType attribute.
- \textit{gnName}: This attribute is filled in by the chassis manager with information found in the chassis slot table when the board is first created.

**Modeling Ports with GnPort**

Port models are very similar to board models, GnSNMPDev provides one port model type that should be sufficient for most modeling needs. The GnPort model type is the default model used to model ports using the GnSNMPDev’s chassis support.

**Port and Board Model Information**

This following information is not necessary for modeling your ports and boards, but is presented to enhance your understanding of how the information for each board and port is read and displayed in OneClick.

All external attributes associated with the boards and ports are read through the application model(s) used to support the board and port models. This is because the application models contain the MIB model types and thus the external attributes which are associated with the boards and ports.

As mentioned previously in this chapter, in OneClick, you can view the ports for a device on the Interfaces tab in the Component Details pane. To view the boards for a device, use the Locater tab to search for the boards by model type name.
Adding Support for Additional Traps

SPECTRUM is able to notify you about significant occurrences on your network through the use of traps (alerts sent from SNMP-compliant devices), events, and alarms.

- An alert is an unsolicited message sent out by a managed node on a network. A more specific definition of an alert depends on the management protocol that is used to report the alert. In general, SPECTRUM uses SNMP as the management protocol to communicate with devices on a network. Alerts that are generated by an SNMP-compliant device are called traps. Traps are received by SPECTRUM and converted to events for further processing.

- An event is an object in SPECTRUM that indicates that something significant has occurred within SPECTRUM itself or within the managed environment. Events always occur in relation to a model. When a managed element on the network generates an alert, this alert is mapped to a SPECTRUM event in the appropriate AlertMap file. The event is then generated and takes on the event code specified in the AlertMap file.

- An alarm is an indication that a user-actionable abnormal condition exists on a model. A model usually detects an abnormal condition when an event occurs, and the EventDisp file indicates that an alarm should be generated.

When you create a new model type, typically you will add support for additional traps, events, and alarms. You can do this using the MIB Tools application and the Event Configuration application in OneClick. The high-level process is as follows:

1. Enable the OneClick preference that allows you to choose whether to install or to export the event and alarm support files created by MIB Tools:
   a. In OneClick, click Preferences on the View menu.
   b. In the Set Preferences dialog, expand the MIB Tools folder in the left pane, and select Show Advanced Map Options.
   c. In the right pane, select Yes.

   Enabling this option allows you to use MIB Tools to export the files that support trap, event, and alarm processing to a user-defined directory, so later you can package the files in your new management module (see Distributing a New Management Module on page 38).

2. Identify the MIB that contains the desired trap definitions.
3. In MIB Tools, import the MIB into the MIB Tools database.
4. Also in MIB Tools, map the traps to events, and specify the events that should generate alarms (and the severity of the alarms).
5. While you are still in the Assign Trap Alarms dialog in MIB Tools, do the following:
   a. Under Advanced Options, select Export Trap Support.
   b. For Starting Event Code, enter the event code for the first trap that you have mapped.
      The event code is a 4-byte integer expressed in hexadecimal format. The first 2 bytes contain the developer ID, and the last 2 bytes identify the event with a unique number. You specify the event code for the first trap, and those for the remainder of the traps are assigned sequentially based on the first.
      **Note:** To easily recognize your custom event codes in OneClick, and to prevent potential conflicts with other SPECTRUM event codes, it is recommended that you use a starting event code that begins with your CA-assigned developer ID.
   c. For Directory, click Browse, navigate to the directory to which to export the event and alarm support files (for example, C:\win32app\<vendor_name>), select the directory, and click Open.

6. Click OK in the Assign Trap Alarms dialog.
   MIB Tools creates the appropriate event and alarm support files and exports them to the directory you specified.

7. In Event Configuration, complete the configuration of the events and alarms.
   For example, you should specify the symptoms, probable causes, and recommended actions for the alarms. These messages are displayed in OneClick when the alarms are generated.
   You might also want to specify additional event processing for one or more events, such as logging the event or using the event to clear an alarm or generate another event using event rules.
   In addition, you can customize the default event message that is displayed in OneClick when the events are generated.

**Note:** For more information on using MIB Tools, see the *Modeling Your IT Infrastructure Administrator Guide (5167)*. For more information on configuring events and alarms, see the *Event Configuration User Guide (5188)*.
Distributing a New Management Module

After you have created new model types and added any necessary customizations, you can use the SPECTRUM Extension Integration (SEI) Toolkit to create a virtual CD (VCD) for distributing the new models types to other SPECTRUM hosts.

The SEI toolkit includes a set of command-line tools for creating required files and for assembling and packaging your extensions into a management module that you can distribute. The toolkit ensures that the management module is compatible with software from CA and other third-party developers. It allows customers to install a module in their existing SPECTRUM environment with minimal installation or integration issues.

**Note:** For in-depth information on working with the SPECTRUM Extension Integration toolkit, see the *SPECTRUM Extension Integration (SEI) Developer Guide (0623)*.
Index

A
alarm • 8, 36
alert • 8, 36

B
boardIndex_Attr • 33

C
CDP • 4
CiscoCDPApp • 4

D
Derivable flag • 16, 34
Derivation points • 25
Desc_Key_Word • 24
Device Identification Manager • 9
Device Type attribute • 9, 21

E
event • 8, 36

G
GnChassis_MF • 33
GnChassisDerPt • 27
GnDevIODerPt • 27
GnModule • 34
GnPort • 35
GnRelayDerPt • 27
GnSNMPAppDerPt • 27

I
importing a MIB • 7
inference handlers • 25
Instantiable flag • 16, 34

M
MIB import • 7
MIB Tools • 7
Model Fragments • 33
model fragments • 25
Model Type Flags • 34

N
No Destroy flag • 16, 34

R
read_next • 28
Required flag • 16, 34

S
sysDescr • 24
sysObjectID • 24
SysOIDVerifyList • 24
System_OID_Verify • 24

U
Unique flag • 16, 34

V
VCD • 38
VendorIDVerifyList • 19, 22
VendorOIDVerifyList • 19, 22
Virtual CD • 38
Visible flag • 16, 34

W
watches • 9