SPECTRUM Concepts

Guide

Document 0647

Este folio es consecutivo en orden alfabético por empresa: 07317

Notice

Este folio es consecutivo en orden alfabético por empresa: 07318

Contents

Preface

This guide explains concepts regarding the SPECTRUM approach to an integrated infrastructure management system. It details both the underlying technology of SPECTRUM, and the terminology used in the rest of the SPECTRUM documents.

This document is a prerequisite for using SPECTRUM's toolkits.

What is in this book

This guide contains the following chapters:

- **Chapter 1:** *Introduction* **-** This chapter gives an overview of SPECTRUM's functionality.
- **Chapter 2:** *The SpectroSERVER* **-** This chapter gives details about the components and functionality of SPECTRUM's main server process.
- **Chapter 3:** *Client Applications* **-** This chapter gives an overview of some SPECTRUM client applications. It details how SpectroGRAPH application represents data from the SpectroSERVER; and also describes the support files used by various client applications.
- **Appendix A:** This chapter describes attributes and relations that are useful to third-party developers.
- **Glossary of Terms**: This chapter defines terminology commonly used in SPECTRUM's documentation.

Text Conventions

The following text conventions are used in this document:

Element	Convention Used	Example

Variables
(The user supplies a value for the variable.)
	Courier and Italic in angle brackets (<>)	Type the following:
		`DISPLAY=<workstation name>:0.0`
		`export display`

The directory where you installed SPECTRUM (The user supplies a value for the variable.)	*<$SPECROOT>*	Navigate to: *<$SPECROOT>*`/app-defaults`
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.	*<$SPECROOT>*`/app-defaults` on Solaris is equivalent to *<$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`
Cross-references	Underlined and hypertext-blue	See "Document Feedback" on page 6.
References to SPECTRUM documents (title and number)	*Italic*	*SPECTRUM Installation Guide (5136)*

Document Feedback

Please send feedback regarding SPECTRUM documents to the following e-mail address:

a) correo electrónico se elimina

Thank you for helping us improve our documentation.

Online Documents

SPECTRUM documents are available online at:

http://support.concord.com/support/secure/products/Spectrum_Doc/

Check this site for the latest updates and additions.

a)correo electrónico se elimina
FUNDAMENTO LEGAL: Artículo 3 fracción II, 18 fracciones I y II, y 19 de la Ley Federal de Transparencia y Acceso a la Información Pública Gubernamental, en relación con el segundo párrafo del Segundo Transitorio y primer párrafo del Cuarto Transitorio de la Ley Federal de Transparencia y Acceso a la Información Pública publicada en el Diario oficial de la Federación el 09 de mayo de 2016, así como en el Lineamiento Trigésimo Octavo, Trigésimo noveno y Cuadragésimo, de los Lineamientos Generales para la Clasificación y Desclasificación de la información de las dependencias y entidades de la Administración Pública Federal publicado en el mismo órgano de difusión el 15 de abril de 2016.
Motivación: Datos personales y/o datos financieros y/o patrimonial

Este folio es consecutivo en orden alfabético por empresa: 07322

Chapter 1: Overview

This chapter gives an overview of SPECTRUM's functionality.

In this chapter:

Prerequisites for Readers

This guide assumes a working knowledge of network management. It also assumes an understanding of SNMP concepts, including agents, MIBs, and traps; and how these concepts are used by SNMP to manage a network environment.

Introduction to SPECTRUM

SPECTRUM is a services and infrastructure management system that monitors the state of managed elements, including devices, applications, host systems, and connections. Status information, such as fault and performance data from these elements, is collected and stored. SPECTRUM constantly analyzes this information to track conditions within the computing infrastructure. If an abnormal condition is detected, it is isolated and the user is alerted. SPECTRUM presents the user with possible causes and solutions to the problem.

SPECTRUM's design is based on the client/server model. The primary server, known as the SpectroSERVER, is responsible for the collection, storage, and processing of data. The SpectroSERVER uses Inductive Modeling Technology to perform these functions. Inductive Modeling Technology combines an object-oriented database with the intelligence of inference handlers. The object-oriented database contains model types that define how a managed element is represented, and models that represent specific managed elements. It also contains relations that define possible associations between model types. Inference handlers provide additional functionality to this system by reacting to events produced by SPECTRUM or managed elements.

The SpectroSERVER stores data in the knowledge base that defines model types, models, and relations in the knowledge base. The SpectroSERVER also polls managed elements and receives alert information from the computing infrastructure. It analyzes and stores this information in the knowledge base, and gives client applications access to this information.

SPECTRUM supports a suite of client applications. Its main client application, SpectroGRAPH, provides the graphical user interface that is used to monitor the network and launch other client applications. SpectroGRAPH's views contain a variety of icons, tables, and graphs that represent the different elements of the network. These graphical components present status information, and provide access to management facilities specific to the managed element they represent. SPECTRUM also has a Web interface that provides users with a series of client applications that display in a Web browser. All information presented by client applications is retrieved from the SpectroSERVER.

Chapter 2: The SpectroSERVER

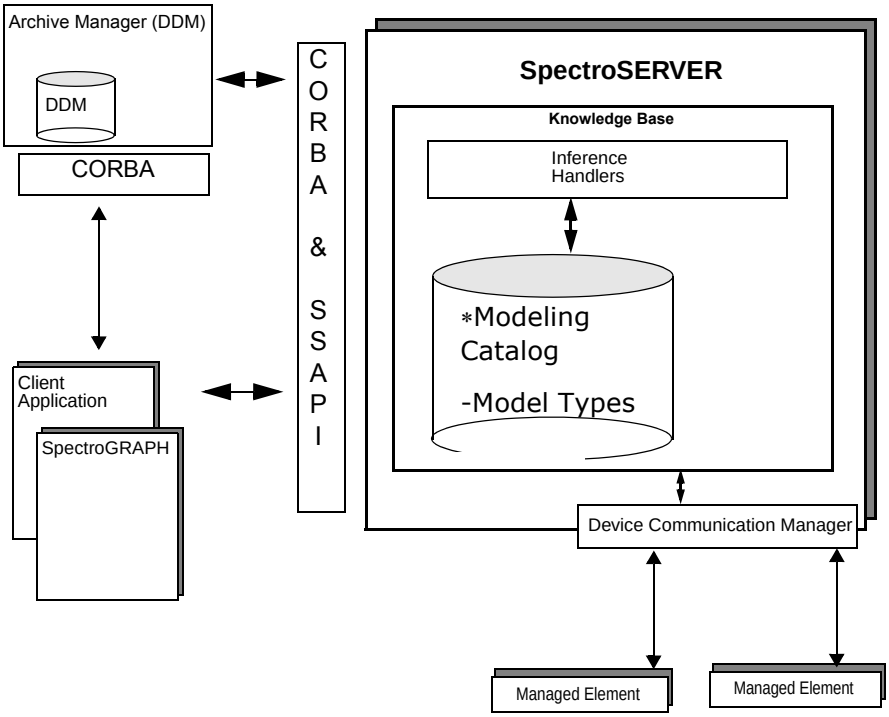
This chapter gives details about the components and functionality of SPECTRUM's main server process.

In this chapter:

An overview of the SpectroSERVER

SpectroSERVER is the primary server for the SPECTRUM product; it functions as a database server, modeling engine, and device manager. SpectroSERVER processes events, generates alarms, and tracks statistics concerning managed elements. All of this information is available to client applications, and can be requested through the SpectroSERVER application programming interface (SSAPI) and the SPECTRUM CORBA interface.

The following illustration shows a simplified view of the various components of SpectroSERVER. The following sections outline these components and the functionality that make up the SpectroSERVER.



Note: SpectroSERVER is also referred to as the **VNM** or Virtual Network Machine. Technically, the term VNM refers to the portion of SpectroSERVER that is responsible for modeling managed elements.

SpectroSERVER operation with threads

SpectroSERVER must handle requests from many client applications, and at the same time access the disk and network. To do this efficiently, SpectroSERVER operates using a multi-threaded architecture that has less overhead than running separate processes. Only one thread executes at any given time; SpectroSERVER maintains a queue of other threads waiting for execution. The SpectroSERVER creates some threads at start-up that terminate only when SpectroSERVER terminates. The SpectroSERVER creates other threads dynamically and terminates them when they are no longer needed. For example, each time a client connects to SpectroSERVER or makes a request through an API, a new thread is started. Normally, it is not necessary to be concerned about SpectroSERVER's internal threading mechanism. However, this concept can become important on a heavily loaded system, when advanced tuning is required to maximize system throughput.

The knowledge base

One of the main components of SpectroSERVER is the knowledge base. The knowledge base is comprised of both the data and the procedural information necessary to manage a computing infrastructure.

The knowledge base has a component that stores model types, relations, models, and event and statistical information. The knowledge base uses a sophisticated system of models and relationships between models to represent and store information about the elements of the computing infrastructure. In essence, this system of models and the relationships between them, when viewed as a single logical entity, describes the computing infrastructure's physical and logical topology to SPECTRUM. SPECTRUM builds its **Root Cause analysis** capabilities on this foundation.

All models in the knowledge base are based on templates known as model types. Model types define the properties that make up an instantiated model. All model types are stored in the knowledge base's modeling catalog.

The knowledge base also contains processes that provide model types with intelligence. These processes include inference handlers and actions. Data generated from or used to support these processes is stored in memory while SpectroSERVER is running and is also part of the knowledge base.

In addition to models and model types, the knowledge base uses the Archive Manager and the Distributed Data Manager to store the historical event and statistical information about specific models. This information is accumulated over time, allowing SPECTRUM to gain extensive knowledge about the computing infrastructure being managed.

The following sections outline each of these components.

The modeling catalog

The modeling catalog is the knowledge base's meta-data. The objects in the modeling catalog ship to the customer, and they are relatively static. However, the customer can manipulate some aspects of the catalog for tuning purposes, or customize it so SPECTRUM can be made aware of new network technologies or new types of managed elements introduced in the computing infrastructure. The following sub-sections describe the various specific types of objects contained within the modeling catalog.

Model types

Model types correspond mainly with different families of managed elements and are the templates used to build models. Model types contain the information (attributes) needed to manage a specific type of managed element. They also possess the intelligence that tells SPECTRUM how the managed element that the model type represents behaves, as well as how the model type should react to events occurring on the managed element, or elsewhere in the network.

For example, the SPECTRUM modeling catalog contains a NokiaFW model type. This model type represents certain types of Nokia Firewalls (e.g., IP330, IP440, IP650, IP740). It is used by SPECTRUM to create a model that represents a specific Nokia Firewall in a customer's network.

Each model type is uniquely identified in the database (the model catalog) using a number (normally represented in hexadecimal format) called a **model type handle**.

Attributes

Each model type has attributes that define much of the declarative knowledge that defines the characteristics and properties of the managed element that the model type represents. These attributes can be either internal or external. External attributes reflect objects from the MIBs supported by the managed element. Internal attributes reflect information that is specific to

SPECTRUM's management of a particular element. All attributes have default values associated with the model type.

In many cases, attributes take on new values when a model of a specific model type is instantiated. The attribute values are specific to the managed element that the model represents. Some attributes, however, are shared attributes. All models of the given model type access the same shared attribute and its value. It is not duplicated in memory or the database for each model.

Each attribute is uniquely identified in the knowledge base using a number (normally represented in hexadecimal format) called an **attribute ID**. There are many attributes that are used across numerous model types. For example, almost every model type in the modeling catalog uses the attribute `Modeltype_name` or `IPAddress`. The attribute IDs for these attributes remain the same across all model types. This normalization of attributes is achieved by using model type inheritance as described in "The model type hierarchy" on page 13.

Relations

Relations define the potential ways that model types can be related to each other. There are many relations defined in the SPECTRUM knowledge base. `Contains`, `Manages`, and `Connects_to` are all examples of relations. Each relation has a unique number (normally represented in hexadecimal format) called a relation handle that identifies it.

Meta-rules

Meta-rules provide meaning to a relation by defining the context in which the relation should be used. A meta-rule identifies the model types that can participate in a relation. To understand the concept of a meta-rule, think of model types and relations in terms of nouns and verbs respectively. Stringing noun and verb phrases together forms a sentence. For a sentence to be meaningful, it must meet three criteria:

- It must be in the format (subject) noun + verb + (object) noun.
- It must be logical; one cannot use any verb to link any two nouns.
- It must reflect reality.

The SPECTRUM notion of meta-rules ensures that the second criterion is met. Meta-rules can be defined on a verb to limit the nouns that the verb can link. Again, meta-rules need to be defined carefully so the restrictions they impose on verbs are logical. Typically, each verb is governed by several meta-rules.

Consider a language in which the following nouns and verbs are defined as the model types and relations that reflect the objects that make up the parts of a computing infrastructure. Further, let's say meta-rules are defined to impose a logic on the way the nouns and verbs can be used together. Meta-rules are defined for a relation and consist of two model types; a left model type and a right model type. This left-right order in meta-rules is the format for building logical sentences; the left model type signifies the subject, the relation signifies the verb, and the right model type signifies the object of the sentence.

Nouns:

"building", "room", "network", "LAN", "printer", "workstation"

Verbs:

"contains", "collects"

Meta-rules:

"contains" [building, room], [room, workstation]

"collects" [LAN, printer], [LAN, workstation], [network, LAN]

To create logical statements in this language, the first two criteria must be met. The following examples meet the first and second requirements and are realistic representations of a computing infrastructure:

"engineering building contains testing lab"

"testing lab contains Pat's workstation"

"Engineering LAN collects Pat's workstation"

"Engineering LAN collects LaserJet printer"

The following are invalid because they do not use the noun/verb/noun format, and therefore do not meet the first criterion:

"contains building collects"

"room LAN workstation"

The following meet the format requirement, but are either illogical or do not follow the defined meta-rules:

"building contains workstation"

"LAN collects room"

"printer collects LAN"

Cardinality of relations

Relations are defined to have a cardinality of either one-to-many or many-to-many. The `Contains` relation, for example, has a one-to-many cardinality. A meta-rule has been defined that allows the `Contains` relation to exist between the Room model type and the Workstation model type. Because it is a one-to-many relation, a single room can contain many workstations, but a single workstation can only be in one room.

The `Connects_to` relation is an example of a many-to-many relation. A meta-rule has been defined that allows the `Connects_to` relation to exist between the switch and router model types. A single switch is connected to many different things (one of them the router). Likewise a router is connected to many things (one of them being a switch).

The value that Cardinality of Relations, is it allows SPECTRUM models to be logically linked, associated or combined in ways that can truly represent the real world computing infrastructure.

The model type hierarchy

Model types are built in a hierarchical fashion, with the more general model types being built first, and the more specific model types being derived from the more general ones. A model type is derived using the principal of inheritance. Multiple inheritance is used to derive a model type from multiple base model types. A derived model type inherits both the attributes and the intelligence of the model type(s) it is derived from. The derived model type also participates in all the meta-rules

that the base model types participate in, and uses the inference handlers that the base model types use. Model types derived from multiple base model types inherit attributes and inference handlers from the base model types in a specific order. This prevents the derived model type from inheriting an attribute or an inference handler multiple times and also determines the initial value of an attribute. New attributes, both internal and external, as well as new inference handlers can be added to the derived model type. The derived model type is a more specific type than its base.

Models

In addition to storing model types, the knowledge base stores all of the models that have been instantiated to represent elements of the computing infrastructure. A model is created by instantiating a specific model type, that is, a copy of the template (the model type) is made, and the copy is then used to represent a real world element in the computing infrastructure.

When a model is instantiated, the attributes of that model type take on values. The knowledge base also stores the current value for each attribute of the model. Some of these model attributes are "shared," being common to all elements of the same type, and describe aspects or behavior of the model type. Each model of a given model type has the same value for these shared attributes. The non-shared attributes have values that may differ for each model based on the current working condition in the infrastructure. The values of the attributes describe the unique aspects, characteristics, and behavior of the single model.

Associations

When SPECTRUM creates a representation of computing infrastructure components using models, these models do not exist as isolated elements. Models relate to one another just as the elements in the computing infrastructure relate to one another. When SPECTRUM instantiates a model, the applicable relations between the model and other models are also instantiated. An instantiated relation is called an association. The association must follow the meta-rules that define the relation. The meta-rules are enforced by SpectroGRAPH and other client applications, not by SpectroSERVER.

For example, consider the relationship between a buyer and seller of a home. The nouns (model types) are BUYER, SELLER, and HOUSE. The verbs BUYS and SELLS capture the potential relationships that can exist between the nouns. These verbs are the relations. The relations need meta-rules to provide meaning. The two meta-rules are BUYER BUYS HOUSE and SELLER SELLS HOUSE. Given these meta-rules, the BUYS and SELLS relation begin to have some meaningful value to the modeling system.

An association would apply a meta-rule to existing models. Assume that BuyerSmith is a model of type BUYER, and SellerJones is a model of type SELLER. NiceNewHouse is a model of type HOUSE. Given the meta-rules, it's valid to set up an association that states BuyerSmith BUYS NiceNewHouse. Likewise, it is possible to set up an association that says SellerJones SELLS NiceNewHouse. However, the meta-rules do not allow a relationship such as NiceNewHome Buys BuyerSmith; therefore this association could not occur.

For further information on how SPECTRUM uses models to represent managed elements, see "Modeling managed elements" on page 16.

Inference handlers

Inference handlers define the behavior and intelligence of a model type. Each inference handler can perform a specific task. The task can be as simple as changing the value of an attribute, or it may be as complex as discovering all the managed elements on a segment of a network. An inference handler may perform a generic task like calculating an average, or it may perform a task specific to a model type, such as creating models of ports in LAN switches. Essentially, inference handlers are the many pieces of intelligence that are the heart of SPECTRUM; without them, SPECTRUM could not offer the infrastructure management capabilities it does.

An inference handler is actually a C++ code segment that is associated with a model type. It is typically a dormant piece of code that supports a wide variety of triggers. Once triggered, an inference handler performs a task. As a result of this task, a new piece of data can be collected, the modeling scheme may be changed, or another SPECTRUM subsystem can be triggered (such as another inference handler). Once the inference handler's processing is complete, it goes into an idle state and awaits another trigger.

Inference handlers specify the behavior of the models of a model type, as well as how the model type reacts to a given set of conditions. They can define:

- The behavior of a model when it is created, destroyed, or activated (a model is activated when it establishes communication with the managed element that it is modeling).
- The behavior of a model if the values of its attributes change, or if an event is generated on it (see "Alerts, events, and alarms" on page 18).
- The behavior of a model if it forms a new association with another model, or is removed from an existing association.
- How certain actions are to be handled (see "Actions" on page 16).

Inference handlers are related to model types within the knowledge base, and they execute on behalf of instantiated models of that model type. When the external condition of two models of a model type changes in a similar way, the reaction of both models is similar. However, the values of the specific model's attributes that reflect the status of that model have an impact on the inference handler. The attribute values of one model may be different the other model's attribute values. Therefore, even though the external condition is the same, and the inference handlers react in a similar way on behalf of each model, the end result of the reaction by the two models of the same model type can be different.

For example, an inference handler associated with a model type that represents a router is designed to perform one specific task: to create models that represent the interfaces of the router whenever a new router model is instantiated. Once this task is complete, the inference handler's job is finished and it waits for the next router model to be created from this model type, so it can perform this job again.

The router model type has attributes that record the number and type of interfaces that exist on the router. Each instantiated router model represents a specific router in the computing infrastructure, and it is highly likely they will have different values for these attributes. The number and type of interface models created by this inference handler are based on these values. Thus, if multiple router models are created in the knowledge base representing different routers of

the type in the network, the same inference handler creates a different (but appropriate) number and type of interface models for each new router model.

The inference handler discussed above may once again be triggered when the router model receives a notification that the real world router has been reconfigured. If the number or type of interfaces on the router has changed as a result of this reconfiguration, the inference handler recreates these interface models based on the new information. This dynamic adaptive modeling capability is an example of one of the fundamental uses of inference handlers throughout SPECTRUM.

Actions

SPECTRUM defines a set of operations that can be performed on a model, such as reading or writing an attribute. To expand on the number of operations that can be performed on a model, SPECTRUM provides a mechanism called an action. Sending an action to a model causes the model type to react in some way; for example, it may return requested data to the action's sender, or it may cause the model type to perform a specific task.

The Archive Manager and the Distributed Data Manager

Each SpectroSERVER has a Distributed Data Manager (DDM) database that is implemented in the Archive Manager server. (The terms Archive Manager and Distributed Data Manager are used interchangeably, and refer to the same set of functionality.) The database stores logged historical events and statistics that relate to specific models, and assists client applications needing to access the stored information. Statistics are the result of collected, logged, and analyzed attribute data linked to a particular model or group of models. Events indicate that something significant has occurred within SPECTRUM itself, or within the managed environment. Event generation is covered in-depth in "Alerts, events, and alarms" on page 18. The sections on Device Communication and Inference Handlers discuss how attribute data is acquired and analyzed.

Summary

The goal of SPECTRUM's knowledge base is to match each real world managed computing infrastructure elements to a pre-defined model type in the modeling catalog. These model types not only hold the information (attributes) that describes the characteristics of the managed element and the knowledge needed to manage a specific type of managed element, but they also possess the intelligence (inference handlers and meta-rules) that tells SPECTRUM how the managed element should behave, as well as how it should react to events happening to it or elsewhere in the network.

This system of intelligent models and relationships, when viewed as a single logical entity, describes the computing infrastructure's topology to SPECTRUM. This knowledge is the foundation of SPECTRUM's patented Root Cause analysis capabilities.

Modeling managed elements

The SpectroSERVER uses models to represent managed elements, and these models are based on the model types defined in the modeling catalog. Some model types can be instantiated to represent a device, an application, or host that operates in the computing infrastructure. The SpectroSERVER can communicate directly with these managed elements using SNMP, if

appropriate. Some model types are instantiated into models that act as containers and are used as a way of grouping other models together. For example, you might create a LAN model to group certain managed elements on a segment of the network, or you might create a Room model to group the managed elements in one room of the building.

A **container model** may contain other container models, models that represent managed elements, or both, depending on the container model type. For example, an IPClassB container model could contain a model that represents a router, and it can also contain several LAN models that represent a range of subnets. However, a Building model can only contain container models; a Floor, a Section, or a Room.

The SpectroSERVER uses **management modules** to manage the specific elements of a computing infrastructure. A management module is made up of model types, relations, inference handlers, and support files. A management module uses a series of models to represent all components of the specific type of managed element. The managed element can be represented with a device model, and the functionality of this device can be supported with a combination of other types of models, such as application models. Each major functional component of a managed element can be modeled as a separate application or can be incorporated into the device model. An application often corresponds to the functionality of a MIB, or a mandatory or optional section of that MIB.

All of the models used to represent a managed element are based on model types defined in the modeling catalog. The associations that the models have with one another are based on the relations and meta-rules defined in the modeling catalog. The SpectroSERVER can implement a variety of associations. For example, a container model can contain models that represent managed elements or container models, connections between device or port models can be established representing physical or logical connectivity, and application models that support a device model express relations showing what functionality a device provides.

> **Note:** Not all model types defined in the SPECTRUM knowledge base can actually be used to create a model in SpectroGRAPH. Some are used only as base model types from which other model types are derived.

Device discovery

SPECTRUM supports an automatic method of discovering and modeling managed elements and connections within the computing infrastructure. This method is called **AutoDiscovery**.

There are two components in AutoDiscovery. The AutoDiscovery application scans IP address ranges or lists, and reads a select group of key MIB objects from each SNMP-enabled managed element encountered. The result can be either presented to the user, or sent to SpectroSERVER for modeling. The server side or back-end AutoDiscovery process uses the key MIB objects of each managed element to determine the best model type to use. It then creates an instance of that model type to represent that managed element.

After models are created and activated for all managed elements found by the AutoDiscovery application, the back-end process determines the placement and connectivity of the models, based in part on user-specified options. Models of bridges and workstations are placed inside a LAN container based on their IP address, and connected to other bridges based on spanning tree and source address tables (as read from the managed elements' MIBs). Models of routers are placed in network containers or in the Universe, and are connected to LAN models or other routers based on IP addresses and masks, IP route table information, or proprietary discovery protocol MIBs.

SPECTRUM also lets the user create specific models from the SpectroGRAPH interface. Two methods can be used to do this. The first way to create a model is by using the IP address or the DNS name of the managed element. With this information, SpectroSERVER contacts the managed element, retrieves the managed element's Name, Vendor, Description, Location, and SysOID, and creates a model using the model type that best represents the functionality of this managed element.

It is also possible to create a model by choosing the model type to base the model on. In this case, you must still provide an IP address or a DNS name so SpectroSERVER can communicate with the managed element. However, the model type you choose is instantiated, regardless of the SpectroSERVER's assessment of the managed element's functionality. All of the appropriate supporting model types are created by SpectroSERVER, and match the functionality described by the managed element's MIB(s).

Device Communication Manager

The Device Communication Manager, or DCM, is the interface between SpectroSERVER and the managed elements. The DCM includes various protocol interfaces that communicate with managed elements using a specific protocol. There is one interface for each of the two supported protocols, SNMP and ICMP. When SpectroSERVER needs to communicate with the managed element, the request is sent on to the appropriate protocol interface in the DCM. The DCM, in turn, passes the request to the managed element.

Polling

SpectroSERVER constantly updates its knowledge of network conditions using polling and logging services. The DCM handles communication with the managed element being polled. When attributes for a model type are defined, they can either be external (to be obtained from the managed element) or internal (stored either in memory or the database). Some external attributes are defined as polled, meaning that SpectroSERVER polls the managed elements on a regular basis. The frequency of the polling is based on the value of the `polling_interval` attribute that is defined for the model. Values of external attributes that do not have the polling flag set are obtained from the managed element whenever a client application or inference handler requests the value.

Logging

Attributes can also be defined as being logged, meaning that their values are written through the Archive Manager to the DDM database. The frequency with which values are logged is based on both the `polling_interval` and the `Poll_Log_Ratio` defined for the model.

For example, if a `polling_interval` of 60 is defined, and the `Poll_Log_Ratio` is set to 10, the attribute value is logged to the statistics file every tenth poll, or every 600 seconds. It is important to note that polling and logging impacts the performance of the SpectroSERVER and the network. Shortening polling intervals and decreasing logging ratios can limit the responsiveness of the SpectroSERVER and generate an unacceptable amount of network traffic.

Alerts, events, and alarms

SPECTRUM is a services and infrastructure management system designed to notify you if there is a fault with a particular managed element in the computing environment. One way that SPECTRUM accomplishes this is by receiving alerts (usually SNMP **traps**) from problem areas in the computing

infrastructure, and converting those alerts into events and alarms to be displayed in various SPECTRUM applications. SPECTRUM uses a series of support files called event configuration files to indicate how alerts, events, and alarms should be processed.

Alerts

An alert is an unsolicited message sent from a managed element to SPECTRUM. The primary management protocol that SPECTRUM uses to communicate with managed elements is SNMP. An alert sent by an SNMP compliant managed element is called a trap. Managed elements with SNMP traps enabled can be configured to direct their traps to the SpectroSERVER. SpectroSERVER uses the trap's source IP address to identify the model associated with that managed element. Once the model is known, the trap is processed as directed by the AlertMap file that is associated with that model type. An AlertMap file exists for most device model types within SPECTRUM. The AlertMap is an ASCII file that is used to map SNMP traps into SPECTRUM events.

Events

An event is an object representing an instantaneous occurrence within SPECTRUM. Events usually indicate that something significant has occurred in relation to the model or other component. Most device model types have an EventDisp event configuration file associated with them. An EventDisp file is an ASCII file that indicates how an event should be processed. After an AlertMap file converts an SNMP trap into an event, the EventDisp file instructs SPECTRUM on how to handle this event for this particular model. The processing of an event may include logging the event and generating an alarm.

Alarms

An alarm is an object that indicates a user-actionable, abnormal condition exists in the managed environment. Usually an alarm is generated when an event has occurred, and the EventDisp file specifies that an alarm should be generated. It is also possible for an alarm to be generated as a result of a watch configured in the SpectroWATCH client application, or as a result of SPECTRUM detecting some sort of abnormal situation not based on an event. When the abnormal condition that caused the alarm ends, the corresponding alarm may be cleared automatically by another event, or by the user. Alarm notifications may be sent to applications and inference handlers that need this information. One of SPECTRUM's strengths is its ability to evaluate a myriad of network events and produce a small number of significant alarms.

Event Format and Probable Cause files help display information concerning events and alarms in client applications. These support files are covered in "Client Application support files" on page 23.

Landscapes and the Distributed SpectroSERVER

A landscape is the SPECTRUM term for a network domain managed by a single SpectroSERVER. A landscape is composed of the models, associations, attribute values, alarms, events, and statistics belonging to a specific SpectroSERVER. Each landscape contained in a network is unique, and must be identified by a unique **landscape handle (ID)**. If desired, each landscape can be represented by a landscape icon in SpectroGRAPH. The landscape icon provides a graphical representation of a SpectroSERVER knowledge base.

Distributed SpectroSERVER (DSS) is a powerful modeling feature that enables the distribution of management for portions of a large-scale network, either geographically, or across multiple servers in a single physical location. DSS can improve SPECTRUM performance when managing a

computing infrastructure by distributing the network load introduced by management traffic, and delegating management functions to remote workstations. Using DSS, you can create a unified representation of the computing infrastructure, composed of multiple landscapes, each with its own local SpectroSERVER. In a DSS environment, a SpectroSERVER client, such as SpectroGRAPH, can access information from more than one SpectroSERVER at the same time. The *Distributed SpectroSERVER (2770)* guide contains tips on how to most efficiently segment your network into landscapes.

When you model multiple landscapes using Distributed SpectroSERVER, the database for each landscape must contain identical modeling catalogs. This means that all model types that exist in one landscape's modeling catalog must also exist in the modeling catalog of every other landscape. If you install new management modules in one landscape, you must install the same management module on every landscape.

Administration of all of the modeling catalogs in a distributed environment is made easier with the concept of a master catalog. The master catalog is the SpectroSERVER that you designate to be used to update the other SpectroSERVERs in the landscape map. When a change is necessary, this change is made to the master catalog. The entire master catalog is manually copied to all other SpectroSERVERs in the landscape map, propagating all changes and keeping the modeling catalogs consistent. See the *Distributed SpectroSERVER (2770)* guide for more information.

Chapter 3: Client Applications

This chapter gives an overview of some SPECTRUM client applications. It details how the SpectroGRAPH application represents data from the SpectroSERVER; and also describes the support files used by various client applications.

In this chapter:

An Overview of client applications

SPECTRUM's main client application is SpectroGRAPH. There are also a number of other client applications that let users interact with the information stored and processed on the SpectroSERVER.

Below is a list of a few of the client applications that may be needed when customizing SPECTRUM or integrating with SPECTRUM.

- **Alarm Manager:** This application is the user interface for alarm information.
- **Alarm Notifier:** This application is used to forward alarm data to user-defined scripts or third-party applications.
- **SANM:** This application is used with the Alarm Notifier to specify policies that filter alarm data sent to user-defined scripts or third-party applications.
- **Event Log:** This application is the user interface for event information.
- **SpectroWATCH:** This application allows the user to establish thresholds and initiate logging for attribute values for specific model types.

The following applications, though not SPECTRUM clients, are required for some SPECTRUM customization and integration.

- **Process daemon:** The process daemon is a process launching and tracking daemon that gives SPECTRUM the ability to control various processes running on a workstation. It starts processes when requested by an application, such as the Control Panel. It can also start processes on

system boot if configured to do so. It automatically restarts critical processes if they stop unexpectedly. The SPECTRUM Control Panel is the only executable actually launched by the SPECTRUM user. All other applications are launched by the process daemon following a request by the user or another application. The process daemon operates in the background and is transparent to the user. It automatically starts during SPECTRUM installation and whenever the system is started.

- **Model Type Editor:** This application is used to derive new model types that support the development of new management modules.
- **JMib Tools:** This application is useful for examining MIBs on managed elements that have SNMP support.

The SpectroGRAPH application

The SpectroGRAPH application displays information from SpectroSERVER using icons and views. Icons denote the models defined to represent the computing infrastructure's managed elements. Views are the various ways in which data from SpectroSERVER is organized for display.

Icons

Icons are graphic representations of instantiated models based on model types from the SPECTRUM modeling catalog. There are many different types of SPECTRUM icons. Icons can represent individual managed elements, groups of managed elements, geographic locations, users, landscapes, connections between models, etc. **Pipes** are a special type of icon used to represent connectivity between managed elements.

General information about a model, such as the model name and model type name, is visible on the icon. Detailed information about a model is found within various icon subviews that are accessed by clicking on double-click zones on the icon. Icons use color to indicate the condition of the managed elements they represent.

The icons available in SpectroGRAPH depend on the SPECTRUM management modules installed.

For information on the support files used to construct icons, see "Client Application support files" on page 23.

Views

The concept of a view in SPECTRUM simply translates into a way of organizing data so it can be viewed or manipulated. There are two main types of views in SPECTRUM, **management views** and **hierarchical views**. Management views focus on various ways to represent data concerning a specific managed element. The hierarchical views represent ways in which the computing infrastructure data can be structured. There are three types of hierarchical views, Topology, Location, and Org-Chart/Services.

The **Topology view** is really an abstraction of the components of the computing infrastructure. When working with this view, you represent the physical or logical components of the computing infrastructure, and group these components with logical connectivity in mind. Pipes that join the icons of the connected models represent connections between managed elements, and between managed elements and other topology containers (such as LANs).

The **Location view** organizes the computing infrastructure's data in terms of physical location and geography. You can start with the global offices and go right to the wiring closets in each of the floors of each of the buildings in each of the regions that the offices are located.

The **Org-Chart/Services view** allows you to group subnets and device models based on organizational considerations such as corporate structure, services, or administrative responsibilities. You can set up your network model to show the departments that rely on various network devices, the devices that provide infrastructure for network services, or the operators that are responsible for sets of network devices. This enables you to prioritize the way in which you troubleshoot network failures.

The type of container models used to group models is based on the type of hierarchical view used. For example, if you use the Topology view, you might create a LAN model to group certain managed elements on a segment of the network. If you use the Location view, you might create a Room model to group the managed elements in one room of the building.

Once a model of a device is created, SPECTRUM communicates with that managed element and gathers information concerning MIB variables, ports, and applications that pertain to the element. This information can then be viewed within one of several different types of management views.

For example, the Device Topology (DevTop) view represents a device in terms of its ports and port connections. You can use the DevTop view to examine existing modeled connections to a device, or to model new connections to other devices. The Application view contains application models that SPECTRUM automatically creates to monitor the functionality of each managed element. Application models are generally abstractions of a MIB, or a mandatory or optional section of that MIB; they rarely represent applications running on the device. Other views include information on things like performance, configuration, and SPECTRUM attribute values. For more information on views, see *Spectrum Views (2517)*.

Client Application support files

SPECTRUM uses several support files to produce the icons, views, and textual information displayed in the user interfaces of the client applications. Most of these files are text files and can be modified with a text editor, or in some cases with a SPECTRUM tool.

Views and icons in SpectroGRAPH

- **CsPib:** These files are Perspective Information Block (PIB) files and are located in `/SG-Support/CsPib`. A CsPib file maps a model type to a CsIib file (see below) for a specific view type, so SpectroGRAPH knows which icon is appropriate for any given model type. A separate CsPib file exists for each view type, including Location, Topology, and Device Topology.
- **CsIib:** These are Icon Information Block (IIB) files and are located in `/SG-Support/CsIib`. There are many different types of CsIib files, and each one plays a role in defining the appearance of an icon. They contain information about the icon size, background image on the icon, and the sub-icons contained within the primary icon. CsIib files also contain the submenu picks available for the icon, and what actions or views launch from these submenu picks. Certain menu picks that are available for all icons are defined in the CsStdMenu file in the `/app-defaults` directory.

- **CsGib:** These are Graphical Information Block (GIB) files and are located in a sub-directory of `/SG-Support/CsGib`. GIB files are related to a particular model type and are contained in a sub-directory of the GIB directory that is named for the model type. There are three different types of GIB files that can be present in each directory:
    - Files with a `.100` extension define the contents of the creation dialog that appears when you select the New Model option to create a model. The role of this GIB file is to allow user input of enough information to uniquely identify an instance of a model type and, for models of devices, to initiate SPECTRUM's management of a represented managed element.
    - The files with a `.30` extension define the contents of a particular GIB view.
    - Files with a `.GTb` extension define tables that are present in a GIB view.

  Use the GIB Editor tool to create and edit GIB files.
- CsImage: These are image files that control the background images available in SpectroGRAPH.

SpectroGRAPH menus

As described in "Views and icons in SpectroGRAPH" on page 23, the CsIib files specify the icon subviews available from a specific icon. Additional menu picks are defined in the CsStdMenu file located in the `/app-defaults` directory. This file manipulates the commands that are available in the File, Edit, View, Tools, and Icon Subview menus in SpectroGRAPH. The entries in these files create commands that are not specific to a certain type of model; they are always available from the SpectroGRAPH menu.

Alarm Manager and Search Manager icons and menus

The icons and menus in the Alarm Manager and Search Manager applications use a different set of support files.

- **mttpl.map:** The mttpl.map file maps model types to a specific icon template. The mttpl.map file is located in the `/SG-Support/CsResource/`Templates directory. If a line entry for the model type does not exist in this file, the default icon is displayed.
- **.tpl files:** These files define the icon templates used in the mttpl.map file. They are located in the `/SG-Support/CsResource/Templates` directory.
- **mm.tpl:** This is a specific .tpl file that keeps track of all management module level icon templates.
- **.fig :** These files store the graphics used on icons. All of these files are located in the `/SG-Support/CsResource /symbol` directory.
- **.isv:** All .isv files are located in the `/SG-Support/CsResouce/actions` directory. Generally, their names represent the management module to which they pertain. Each line of an .isv file corresponds with a menu entry for the model type.
- **isv.map:** This file maps .isv files to specific model types. This file is also located in the `/SG-Support/CsResource/actions` directory. The isv.map file contains a listing of .isv files referenced by model type handle and model type name.

Event and alarm text

EventDisp files that exist on the SpectroSERVER file system in the `/SS/CsVendor/<vendor name>` directories define how events will be processed by SPECTRUM. When processed, some events generate alarms. Data about events and alarms can be viewed in various client applications. There are two types of support files that contain the text that describes a specific event or alarm in the client applications:

- **CsEvFormat:** There is a specific CsEvFormat file, also known as an Event Format file, for each type of event generated by SPECTRUM. These files define the event text presented in the Alarm Manager and Event Log applications. If an alert from the network generated this event, variables representing alert data may be present in this file. CsEvFormat files are located in the `/SG-Support/CsEvFormat` directory.
- **CsPcause:** There is a specific CsPcause file, also known as a Probable Cause file, for each alarm generated by SPECTRUM. These files define the probable cause text presented in the Alarm Manager. CsPcause files are located in the `/SG-Support/CsPcause` directory.

Appendix A: Attribute and Relation Definitions

This chapter defines the attributes and relations that are used or referenced by one or more of SPECTRUM's integration points.

In this chapter:

Attributes

The concept of an attribute and its role with respect to models and model types is outlined in "Attributes" on page 11.

Many attributes contained in the knowledge base are counterparts to MIB variables. External attributes directly correspond to specific MIB variables. Many internal attribute values are based on MIB variables values and have undergone some mathematical calculations to arrive at their SPECTRUM value. The naming conventions used for these attributes usually make the linkage between the attribute and the MIB variable obvious.

The following table shows the attributes that have been defined to help you integrate other applications with SPECTRUM. These attributes are when creating a new management module, when using the Southbound Gateway toolkit, or when using the Modeling Gateway toolkit. This table is a quick reference with attributes grouped by functionality. Following this table is an alphabetical listing of the attributes with in-depth definitions.

Application Model Discovery		
Attribute	**Attribute ID**	**Found On**
default_attr	0230006	Application model types
Device Model Discovery		
Attribute	**Attribute ID**	**Found On**

DeviceNameList	0x1293E	Device model types
DeviceType	0x23000e	Device model types
Disposable_Precedence	0x114e2	Device model types
Enable_IH_Spec_Dev_Name	0x3d0062	Device model types
Enable_IH_Device_Name	0x3d0001	Device model types
Image_Index	0x3d0001	Device model types
System_OID_Verify	0x110bb	Device model types
System_OID_Verify_List	0x12910	Device model types
General Model Type Information		
Attribute	**Attribute ID**	**Found On**
CompanyName	0x118b8	Device and application model types
Description	0x230017 and 0x118bc	Device, application, and interface model types
DeviceType	0x23000e	Device model types
Manufacturer	0x10032	Device model types
MMName	0x1196a	Device and application model types
MMPartNumber	0x1196b	Device, application, and interface model types
Model_Class	0x11ee8	Device model types
Model_Name	0x1006e	Device, application, and interface model types
Modeltype_Name	0x10000	Device, application, and interface model types
Vendor_Name	0x11570	Device, application, and interface model types
Network Information		
Attribute	**Attribute ID**	**Found On**
Network_Address	0x1027f	Device, application, and interface model types
Network_Mask	0x110b8	Device, application, and interface model types
Polling Information		
Attribute	**Attribute ID**	**Found On**
polling_interval	0x10071	Device, application, and interface model types

poll_log_ratio	ox10072	Device, application, and interface model types
pollingstatus	0x1154f	Device, application, and interface model types
Port Identification		
Attribute	**Attribute ID**	**Found On**
ifAlias	0x11f84	Interface model types
if_Index	0x11348	Interface model types
ifName	0x11f60	Interface model types
if_Phys_Addr	0xd0399	Interface model types
ip_address	0x1196b	Interface model types
SNMP Information		
Attribute	**Attribute ID**	**Found On**
Community_Namd	0x10024	Device, application, and interface model types
CommunityNameForSNMPSets	0x11a7f	Device, application, and interface model types
isManaged	0x1295d	Device model types
Security_String	0x10009	Device, application, and interface model types

The following section gives detailed definitions of the attributes outlined in the previous table. Each attribute definition includes:

- The attribute ID.
- The data type of the attribute value (can be scalar or list). The possible data types are:
    - Boolean
    - Integer
    - Enumeration
    - Real
    - Date
    - Time Ticks
    - Counter
    - Counter64
    - Gauge
    - Model Handle, Model Type Handle, Relation Handle, or Landscape Handle

  - Attr ID
  - Octet String
  - Text String
  - Object ID
  - IP Addr
  - Agent ID
  - Group ID
  - Tagged Octet
  - Unsigned Integer
- A basic description of the attribute.
- A list of some of the enumerated values for the attribute (if applicable). *For a complete and up-to-date list of enumerated values, check the Model Type Editor.*
- There are several attribute flags that can be set for each attribute. The attribute definitions below indicate which of the following four main attribute flags are set to `TRUE`.
  - **External:** Indicates that the value for this attribute is maintained outside of SpectroSERVER and that an update of the attribute value is done either at a user request or at a polling interval.
  - **Readable:** Informs SpectroSERVER that a client or other application is allowed to read this attribute value from SpectroSERVER. If the External flag is set, set this flag in accordance with the MIB definition of the Readable variable for this attribute. If the External flag is not set, set this flag as desired.
  - **Writable:** Informs SpectroSERVER that a client or other application is allowed to write this attribute value to the SpectroSERVER database. If the External flag is set, set this flag in accordance with the MIB definition of the variable for this attribute. If the External flag is not set, set this flag as desired.
  - **Shared:** Declares that only one value exists for this attribute and that all models of the current model type share the same value. The value is not duplicated in memory or the database for each model.

Community_Name

Attribute ID: 0x10024

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute identifies the community string used when SPECTRUM attempts to communicate with a managed element using SNMP. This attribute is evaluated when performing SNMP get(s). If the attribute `CommunityNameForSNMPSets` is empty, this attribute is used when performing SNMP set(s), as well.

CommunityNameForSNMPSets

Attribute ID: 0x11a7f

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute specifies which community name is used when performing SNMP sets. If left blank, the `Community_Name` attribute value is used for SNMP sets.

CompanyName

Attribute ID: 0x118b8

Data Type: Text String

Flags: Readable, Shared

Description: This attribute is generally used by device and application model types. It is set equal to the name of the company that developed the model type.

default_attr

Attribute ID: 0x230006

Data Type: Attr ID

Flags: Readable, Writeable, Shared

Description: This attribute is used in the application discovery process to identify the applications that a particular managed element supports. The value of the `default_attr` is set equal to the attribute ID of an attribute that represents a MIB object that uniquely identifies the MIB.

Description

Attribute ID: 0x230017 and 0x118bc

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute provides a textual description of the model type.

DeviceNameList

Attribute ID: 0x1293E

Data Type: Text String (List of)

Flags: Readable, Writeable, Shared

Description: This attribute is populated with device names that correspond to the OIDs in the `SysOIDVerifyList` attribute. To use this attribute to be used to set the device name, set the `Enable_IH_Device_Name` attribute and the `Enable_IH_Spec_Dev_Name` attribute to `TRUE`.

DeviceType

Attribute ID: 0x23000e

Data Type: Text String

Flags: Readable, Writeable

Description: If the model type represents one specific type of device, use this attribute to hold the name of the device type. The value of this attribute is displayed in the field at the bottom of a model's icon. If there are multiple device types for a given device, use the `DeviceNameList` attribute.

Disposable_Precedence

Attribute ID: 0x114e2

Data Type: Integer

Flags: Readable, Writeable, Shared

Description: This attribute is used by SPECTRUM's device discovery mechanism to resolve conflicts in the device model type selection process. If more than one model type has a `System_OID_Verify` value that matches the `SystemObjectID` of the managed element, the model with the highest `Disposable_Precedence` value is selected.

Enable_IH_Spec_Dev_Name

Attribute ID: 0x3d0062

Data Type: Boolean

Flags: Readable, Writeable

Description: When this attribute is set to `TRUE`, SPECTRUM uses the `Enable_IH_Device_Name` inference handler to determine the vendor name using the enterprise number from the device.

Enable_IH_Device_Name

Attribute ID: 0x3d0008

Data Type: Boolean

Flags: Readable, Writeable

Description: When this attribute is set to `TRUE`, SPECTRUM uses the `Enable_IH_Spec_Dev_Name` inference handler to read the device's System Object ID to determine the exact product name.

ifAlias

Attribute ID: 0x11f84

Data Type: Octet String

Flags: Readable, Writeable

Description: This attribute corresponds with this value from the MIB II Interface table.

ifIndex

Attribute ID: 0x11348

Data Type: Integer

Flags: Readable

Description: This attribute corresponds with this value from the MIB II Interface table.

ifName

Attribute ID: 0x11f60

Data Type: Text String

Flags: Readable

Description: This attribute corresponds with this value from the MIB II Interface table.

if_Phys_Addr

Attribute ID: 0xd0399

Data Type: Octet String

Flags: External, Readable

Description: This attribute corresponds with this value from the MIB II Interface table.

Image_Index

Attribute ID: 0x3d0001

Data Type: Integer

Flags: Readable, Writeable

Description: This value links a GnSNMPDev model with an image to be placed on the icon that represents the model. Possible values and their corresponding images are:

Value	Image
1	Generic Device
2	Bridge
3	Router
4	Hub
5	PC
6	Terminal Server
7	Workstation

8	Switch

ip_address

Attribute ID: 0x10e43

Data Type: Agent ID

Flags: Readable, Writeable

Description: This is the IP address associated with an interface model.

isManaged

Attribute ID: 0x1295d

Data Type: Boolean

Flags: Readable, Writeable

Description: This attribute is typically used by device models. When it is set to `TRUE`, SPECTRUM manages this device using SNMP communication.

Manufacturer

Attribute ID:0x00010032

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute is generally used with a device model and shows the manufacturer responsible for the device.

MMName

Attribute ID: 0x1196a

Data Type: Text String

Flags: Readable, Shared

Description: The attribute holds the management module name for device and application models.

MMPartNumber

Attribute ID: 0x1196b

Data Type: Text String

Flags: Readable, Shared

Description: This attribute is used by most device, application, and interface models and shows the part number assigned to the management module by the management module developer.

Model _Class

Attribute ID: 0x00011ee8

Data Type: Integer

Flags: Readable, Writable

Description: The model class defines the type of device the model represents. The following list shows SPECTRUM's model classes and their respective integer identifier.

Model Class	Identifier
Unknown	0
Other	1
Switch	2
Router	3
Switch-Router	4
Hub	5
Switch	6
Link	7
Network	8
WorkStation-Server	9
Container	10
Chassis	11
Pingable	12
MAC	13
SNMP	14
Port	15
User	16
Application	17
Component	18

Landscape	19
ROUTER_APP	20
SWITCH_APP	21
BRIDGE_APP	22
MIB_APP	23
RMON_APPL	24
UNIX	25
NT	26
Firewalls	27
IDS	28
Security_Scanners	29
Anti-Virus_Applications	30
PKI_Systems	31
Packet_Sniffers	32
Syslogs	33
Generic_TL1_ Device	37
VOIP	38
CMTS	39
Wireless	40
Cable_Modem-MTA	41
VPN	42
DSL	43
Multiplexor	44
SAN	45
PBX	46
USER_CUSTOMIZATION	47
PRINTER	48
TRANSPORT_DEVICE	49
SERVICE_MGT_COMPONENT	50
SLA_COMPONENT	51
CUSTOMER	52
DIAGNOSTIC_SCRIPT	53
DIAGNOSTIC_DATA	54
DIAGNOSTIC_COMPONENT	55

HOST_CONFIGURATION	56
Power Supply	103
Amplifier	104
Line Monitor	105
Test Point	106
FIBER_NODE	107
HEFIBER	108
IP_PHONE	109

Model_Name

Attribute ID: 0x1006e

Data Type: Text String

Flags: Readable, Writeable

Description: This is the name given to that model. A model's name helps distinguish it from other models of that model type.

Modeltype_Name

Attribute ID: 0x10000

Data Type: Text String

Flags: Readable, Shared

Description: The textual name of the model type.

Network_Address

Attribute ID:0x1027f

Data Type: Agent ID

Flags: Readable, Writeable

Description: This attribute contains the device model's network address that SPECTRUM uses to identify the model to communicate with. This value is usually an IP address.

Network_Mask

Attribute ID: 0x110b8

Data Type: Agent ID

Flags: Readable, Writeable

Description: This attribute further identifies where a device model is logically located on the network.

polling_interval

Attribute ID: 0x10071

Data Type: Time Ticks

Flags: Readable, Writeable

Description: This attribute is used by most device, application and interface models. The value of this attribute identifies the number of seconds that must pass between polling requests. This attribute can be set by the user in the Model Information view.

poll_log_ratio

Attribute ID: 0x10072

Data Type: Integer

Flags: Readable, Writeable

Description: This attribute is generally found on most device, application and interface models. The value of this attribute identifies the polling cycle that logs the polled attributes. For example, a value of 10 means that the polled attributes are logged every 10th polling cycle.

pollingstatus

Attribute ID: 0x1154f

Data Type: Boolean

Flags: Readable, Writeable

Description: This attribute is used by most device, application, and interface models. This attribute gives the user the ability to enable/disable polling. A value of `TRUE` means that the model should poll. A value of `FALSE` means that the model should not poll.

Security_String

Attribute ID: 0x10009

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute is used for most application, device, and interface models. The value of this attribute defines the community strings that have access to this model.

System_OID_Verify

Attribute ID: 0x110bb

Data Type: Object ID

Flags: Readable, Writeable, Shared

Description: This attribute is used by SPECTRUM's device discovery mechanism to determine the appropriate model type to represent a device. This attribute contains a System Object ID value. When SPECTRUM instantiates a model to represent a managed element, it identifies the appropriate model type by matching the managed element's System Object ID to the model type's `System_OID_Verify` value. If multiple model types are found to have a matching value, the model with the highest `Disposable_Precedence` value is used. If a model type is used to represent a family of devices then the `SysOIDVerifyList` attribute is used instead of the `System_OID_Verify` attribute.

SysOIDVerifyList

Attribute ID: 0x12910

Data Type: Object ID (List of)

Flags: Readable, Writeable, Shared

Description: This attribute is used by SPECTRUM's device discovery mechanism to determine the appropriate model type to represent a device. This attribute contains a list of System Object ID values. When SPECTRUM instantiates a model to represent a managed element, it identifies the appropriate model type by matching the managed element's System Object ID to a value from the model type's `SysOIDVerifyList` list of values. If multiple model types are found to have a matching value, the model with the highest `Disposable_Precedence` value is used.

Vendor_Name

Attribute ID: 0x11570

Data Type: Text String

Flags: Readable, Writeable

Description: This attribute is used by most device, application, and interface models. It identifies the vendor of the managed element.

Relations

The concept of a relation is explained in "Relations" on page 12. The following section defines the relations that are core to SPECTRUM. Each definition below includes:

- The name of the relation.
- The type of relation.
- A description of the relation.

Connects_to

Type: Many to Many

Description: The `Connects_to` relation forms a connection between two models. `Connects_to` causes a connection to be formed between a port model and a device or topology model.

Contains

Type: One to Many

Description: The `Contains` relation allows a model to contain other models. Location models use the `Contains` relation to determine what location or device models can be contained within the location model. Rules for this relation are checked by SpectroGRAPH when attempts are made to add or copy models to Location models.

HASPART

Type: Many to Many

Description: The `HASPART` relation establishes an association between a device model and the models that represent the components of the device. Typically, these components are the device's interface models. However, this relation can also establish an association between a component model and its components.

Links_with

Type: One to Many

Description: The `Links_with` relation is used to represent a resolved connection between two models. This relation is usually found between two port models. Both the Link view and Live Pipes rely heavily on this relation.

Manages

Type: One to Many

Description: The `Manages` relation forms an association between an application model and the device model that is running it. The `Manages` relation can also form an association between an element management system model and the element models of this system.

Provides

Type: One to Many

Description: The `Provides` relation identifies which applications provide which sub-applications. It can also be used to determine which application provided the sub-application in question.

Index

K

L

M

O

P

R

S

T

V

Glossary of Terms

Action

Any operation that is not part of the basic set of operations defined by SPECTRUM for use with a model.

Agent

See "Network management agent" on page 48.

Alarm

An indication that an abnormal condition exists in reference to a model.

Alarm Severity

A value found in the EventDisp file that indicates the model's condition. Conditions represent the presence and seriousness of an alarm. Valid values are 0 through 6 and all represent a different colored condition.

Alert

An unsolicited message sent from a managed element to the SpectroSERVER.

Alert Code

The string that identifies the alert. An alert received from an SNMP source has an alert code that consists of a generic trap followed by a dot followed by an enterprise specific trap.

AlertMap File

An AlertMap file exists for each model type. This file maps incoming SNMP trap data to SPECTRUM events.

API

See "Application programming interface (API)" on page 43.

Application programming interface (API)

A set of routines used to make calls to another software package.

Association

A link formed between two models by a relation.

Asynchronous call

A call to a method that begins, but does not necessarily complete, the requested operation before allowing the program to continue execution. At some point in time, the requested operation completes and notifies the program. In the meantime, the program and the requested operation can both proceed at the same time. See also "Synchronous call" on page 51.

Attributes

The declarative knowledge in SPECTRUM that defines what a model type is. Attributes are defined using the Model Type Editor.

Client

The application process in a client-server architecture. See also "Client-server architecture" on page 44.

Client-server architecture

A system design based on the relationship between a process that provides services, referred to as the server, and an application process that uses the services, referred to as the client.

CORBA

(Common Object Request Broker Architecture) CORBA is a software component architecture, produced by a consortium of over 800 software companies called the Object Management Group (OMG), which enables software components written in different languages (C, C++, Java, Smalltalk, COBOL, ADA, Lisp, Perl, Tcl, Eiffel, Python), resident on different machines, and written for different operating systems to communicate.

Custom Installation Script

A script that can be included in a VCD that will execute as the integration package is being installed on the SpectroSERVER.

Database

A collection of interrelated data organized to facilitate efficient and accurate inquiry and update.

Database management system

A software package that organizes and maintains a database.

DCM

See "Device Communications Manager (DCM)" on page 45.

Developer ID

SPECTRUM uses Developer IDs to ensure that the objects, such as model types, attributes, or relations, created by users or integrators have unique identifiers and can therefore be distributed to other users without conflict.

To obtain a Developer ID from Aprisma, call the Aprisma Technical Assistance Center at 603-334-2978. To be issued a Developer ID, you must have purchased the Level One toolkit. See the *Integrator Guide (5068)* for more information on the Level One toolkit.

A Developer ID consists of a 14 character developer name and a 4 character prefix. The developer name must be alpha numeric and can include underscores (no other punctuation marks are

allowed). The prefix must also be alpha numeric, however no underscores or other punctuation marks are allowed.

Device

A managed element of some kind.

Device Communications Manager (DCM)

A multi-protocol communication engine in the SpectroSERVER that handles communication with all managed elements, regardless of their protocol. The DCM translates SpectroSERVER requests into protocol understood by the individual devices.

Distributed SpectroSERVER (DSS)

A modeling feature that uses the concept of landscapes to improve SPECTRUM performance when managing a large computing infrastructure by distributing the load introduced by management traffic and allowing you to delegate management functions to remote workstations.

Edit Mode

Allows editing the current view in SPECTRUM. Selecting the Edit Mode displays the File and Edit options in the menu bar.

Element Management System (EMS)

A system that enables you to provision, manage, and/or monitor elements in a computing infrastructure.

Event

A significant message from the SpectroSERVER.

EventAdmin Model Type

The Southbound Gateway model type that represents a third-party management system.

Event Code

A hexadecimal number that uniquely identifies an event.

Event Data Template

A series of integers used by Southbound Gateway to format variable data coming from an alert.

EventDisp File

An EventDisp file exists for each model type. This file determines how the event will be processed by SPECTRUM.

Event Format File

This file provides the text for the event messages contained in the Alarm Manager and the Event Notifier.

EventModel Model Type

A model type within SPECTRUM that represents a unique alert source within a Southbound Gateway integration.

Event Severity

A numeric value found in an EventDisp file entry that describes the seriousness of an event. Valid values are from 0 through 100, with 0 being the least severe.

Event Variable ID

An ID that represents the event variable data. Use of this ID will return the event variable value.

Generic Information Block (GIB)

Parameters for controlling generic screen views in SPECTRUM. See also "GIB View" on page 46.

Generic view

See "GIB View" on page 46.

GIB

See "Generic Information Block (GIB)" on page 46.

GIB File

Contains the parameter templates used to display a Generic View.

GIB View

SPECTRUM views, referred to as Generic Views, that use templates to determine what kind of information to present to a user, and the format to use in presenting it. These templates are stored in a GIB file. A GIB view displays a model's configuration, diagnostic, and performance information.

Icon

A graphic or picture representation displayed on a screen.

Icon Information Block (IIB)

Used in SPECTRUM to describe how an icon is to be displayed.

IIB

See "Icon Information Block (IIB)" on page 46.

ICMP

Internet Control Message Protocol. ICMP supports packets that contain error, control and informational messages.

Index File

A file created using the SEI toolkit. This file defines the components of the integration, where they exist on the machine and where these components will be installed on the customer's machine.

Inductive Modeling Technology (IMT)

Aprisma's set of artificial intelligence techniques that allow a computing infrastructure of arbitrary complexity to be modeled such that every element is given intelligence.

Inference Handler

Inference handlers are the intelligence behind SPECTRUM and monitor the changes in a model's environment, as well as changes between models that are related to each other. Inference handlers also monitor the changes in relationships between models and attributes.

Instance Variable

The instance variable ID stores the instance portion of the OID. If the variable binding identifies a particular object from a table variable within the trap MIB, it will likely include an instance ID.

Instance Variable ID

The integer that identifies the instance variable in the OID map. Use of this ID will return the instance variable.

Instantiate

In Object Oriented Design, creating a particular occurrence of something.

Intelligence circuit

A collection of inference handlers that defines the behavior of a model type.

Knowledge base

Everything that can be modeled and managed by SPECTRUM, including concepts, relationships, declarative knowledge and procedural knowledge.

Knowledge base management system

The software products used to define and manage the information in the knowledge base.

Landscape

All data specific to any one virtual network machine (VNM) in a single network.

Managed element

An item or device whose status is being monitored and controlled. A managed element can be a network device, host system, application, service or other computing infrastructure component.

Managed object

A variable on a managed element containing one piece of information about the node. Each node may have several objects.

Management agent

An implementation of a management protocol that exchanges the managed element's information with the management station.

Management Station Access Provider (MSAP)

In SPECTRUM, a software task that provides an object with access to a management station.

Meta-rules

Meta rules specify the model types that can participate in a relation.

MIB

Management Information Block

mkmm

A tool that is a part of the SEI Toolkit. This tool creates the VCD out of the index file and the applicable component files.

mkcd

A tool that is a part of the SEI Toolkit. This tool finished the VCD adding a version number and making the VCD installable on a SPECTRUM host.

Model

Collection of information that forms a specific occurrence of some basic defined type. In SPECTRUM, models are instances of model types.

Model type

A template that describes the attributes, actions, and associations related to a managed element in SPECTRUM.

Model Type Editor (MTE)

The primary tool that defines the concepts, relationships, meta-rules, and declarative knowledge in the SPECTRUM knowledge base.

Modular Object Oriented Threads (MOOT)

A task control manager used in the SpectroSERVER, developed at Aprisma.

MOOT

See "Modular Object Oriented Threads (MOOT)" on page 48.

Motif Window Manager (Mwm)

The window manager shipped with OSF/Motif™, which provides a unique "look and feel" to windows developed with that software.

MSAP

See "Management Station Access Provider (MSAP)" on page 47.

MTE

See "Model Type Editor (MTE)" on page 48.

Navigation Mode

Moving from one view to the next in SPECTRUM. The menu bar contains the File and View options when in Navigation Mode. Navigation Mode does not allow editing of a view.

Network management agent

An implementation of a management protocol that exchanges the managed element's information with the management station.

Network management protocol

The means by which the management station and the managed elements exchange information.

Network management station

The host system or workstation that is running the network management applications and protocol.

Object-Oriented Design (OOD)

A design encompassing the process of breaking a system into parts, each of which represents some class or object from the problem domain, and applied by viewing the world as a collection of objects that cooperate with one another to achieve some desired functionality. Typically includes a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

OID

An identifier for a managed object.

OID Map

The syntax to map an alert variable to an event variable.

OOD

See "Object-Oriented Design (OOD)" on page 49.

OSF

See "Open Software Foundation (OSF)" on page 49.

Open Software Foundation (OSF)

A consortium of industry leaders that have united to direct, contribute to, and fund the continued development of the X Window System as an industry standard windowing system.

Perspective Information Block (PIB)

Information that maps model types to icon images in IIB files in SPECTRUM.

PIB

See "Perspective Information Block (PIB)" on page 49.

Pipe

An icon that represents a connection between two devices or ports.

Pollable attributes

Attribute for which the VNM regularly queries the managed element to obtain current values. Attributes are defined as pollable or non-pollable through the Model Type Editor.

Probable Cause File

This file provides text that describes the probable cause of the alarm. This text appears in the Alarm Manger.

Procedural knowledge

In SPECTRUM, information defining how a concept behaves or reacts to environmental changes.

Protocol

A set of rules used by computers to communicate with each other.

Relation

Information describing the semantic connection that models have with each other.

Report Information Block (RIB)

An external file used for defining report formats.

RIB

See "Report Information Block (RIB)" on page 50.

SEI Toolkit

(SPECTRUM Extensions Integration toolkit) A set of tools that allow you to package and distribute an integration so that it can be installed on other SPECTRUM host machines.

Server

A process that provides services in response to a client request in a client-server architecture. See also "Client-server architecture" on page 44.

SNMP

(Simple Network Management Protocol) A network protocol used to monitor devices and applications on a network.

SNMP Trap

An extraordinary occurrence that is either broadcast or directed to a network management application notifying the application of device or network problems. Traps are generated by SNMP enabled devices or applications.

SpectroGRAPH

Graphical user interface software in SPECTRUM that provides a means for the user to view, edit and interact with the information provided by the system through the SpectroSERVER. A client in the client-server relationship with the SpectroSERVER.

SpectroSERVER

Software in SPECTRUM that controls communication with the database and the managed elements, and acts as an interface between applications, such as the user interface, and the information provided by the database and the devices. Major components within the SpectroSERVER include the Device Communications Manager and the Virtual Network Machine.

SpectroSERVER API

Provides a means for client applications to access information in the SpectroSERVER. See also "Application programming interface (API)" on page 43.

SPECTRUM

Aprisma's services and infrastructure management system.

SSORB Client

An application that accesses SPECTRUM's CORBA interface, SSORB.

Synchronous call

A call to a method that performs the entire requested action before a program can continue running. The program continues only after the completion of the called method. See also "Asynchronous call" on page 44l.

Timing interval

The frequency with which the current view updates displayed model attribute information. The default is one request per model every five seconds for some number of attributes. This can be modified by the user for a generic view.

UI (User Interface)

See "SpectroGRAPH" on page 50.

User Interface (UI)

See "SpectroGRAPH" on page 50.

Value Variable ID

An ID that is used to retrieve the value of a variable binding sent with an SNMP trap.

Variable Bindings

Variable data that is sent as a part of an SNMP trap.

VCD (Virtual CD)

This is a series of files created by the SEI toolkit to allow developers to easily package and distribute their integrations.

View

One of many representations of the network landscape.

VNM

See "Virtual Network Machine" on page 51.

Virtual Network Machine

Within the SpectroSERVER, the software level that provides access to data regardless of where the data is stored. It can be stored in the database, the VNM's memory, or any of the managed elements on the network. The VNM also embodies the SPECTRUM intelligence, known as the Inductive Modeling Technology.

Window

A region on the display created by a client.

Window manager

A client that allows the user to move, resize, circulate, and iconify windows on a display. The window manager used largely determines the "look and feel" of the X system on a particular system.

X

Abbreviation for X Window System.

X Window System

Network-based graphical windowing system.